THE BEST SELLING COMPUTER PROJECTS MAGAZINE OCTOBER 1983

ELECTRONICS & COMPUTING MONTHLY

AN EMAP PUBLICATION

DM 5·80 $2·95 **85p**

5 ORIC PLUG
13 ORIC MONITOR
25 PROGRAMMER
35
47 ORIC PER 1
55 CAT E14
56
66
72 BOOKS
73 ORIC TAPES
82
86
92 MAY 2x PER ORIC
94 ORIC CLUB TAPE PRINTER

★ FREE ★ INSIDE– COMPUTER DATA CARD

BBC EPROM PROGRAMMER

COMPUTER GRAPHIC TECHNIQUES • IN PRAISE OF ZX BASIC

SPEECH RECOGNITION SYSTEMS

WIN A TRIP TO THE LAS VEGAS CONSUMER ELECTRONICS SHOW

Electronics & Computing Monthly
Scriptor Court,
155 Farringdon Road,
London,
EC1R 3AD

EDITORIAL
01-833-0846
Editor
Gary Evans

Editorial Assistant
Ann Houghton

Administration
Liz Gregory

ADVERTISING
01-833-0531

Manager
Claire Fullerton

Deputy Manager
Alun Evans

Classified
Nik Saha

PRODUCTION
01-833-0846
Design
Pat Haylock

Make-up
Time Graphics

Publisher
Alfred Rolington

Distribution
EMAP
National Publications

Published By
EMAP Business And
Computer Publications

Printed by
EMLP Peterborough,
England

Subscriptions
Electronics &
Computing Monthly,
(Subscription
Department),
Competition House,
Farmdon Road,
Market Harborough,
Leicestershire.

ABC
MEMBER OF THE AUDIT
BUREAU OF CIRCULATION

Vol. 3

IN THIS ISSUE

No. 10

See Page 76 for details of our free to enter competition – first prize a trip to Las Vegas.

Electronics & Computing Monthly is normally published on the 13th day of each month

EDITORIAL

After much advance publicity and product reviews in the computing press, Acorn's Electron finally saw the light of day at the Acorn User Exhibition in late August.

The Electron is certainly a good product (see our review starting on page 21) and sales of the machine are almost certain to place a strain on Acorn's production lines for the next few months. Apart from the fact that the Electron is a well made machine selling at a fairly attractive price and its BASIC is by and large compatible with that of the BBC micro, sales of the Electron are bound to be buoyant by virtue of the fact that there will be a wealth of software and is/will soon be available for the machine. Reviews of new computers in *E&CM* continually point out that any computer with designs on the home computer market must be supported by a sufficient variety of software if it is to have any chance of success.

Which Comes First

There is a certain amount of 'chicken and egg' about the situation as it is easy to see that software houses may well be reluctant to expend the time and effort that goes into developing new software unless they can be sure of the success of the target market.

In the Electron's case there are no such problems as, with the exception of programs relying on the BBC's teletext mode, existing BBC software is easily connected to run on the Electron.

The design of the *E&CM* computer also paid heed to the necessity of providing software support for any computer. The approach adopted was to design a system capable of supporting the FLEX OS. There is a wealth of software written to run on FLEX systems and thus when complete, the *E&CM* hi-res computer can make use of 'off-the-shelf' software to perform the majority of talks demanded of the system. (Watch out for a new series detailing the renage of FLEX compatible software available in the UK).

The need for software support extends even to development systems. We have a 16 bit microcomputer development system which, when we publish the design, will come complete with a cross-assembler that will allow the board to run software written for a range of 8 bit MPU's. The board won't make full advantage of a 16 bit MPU's speed but it will be capable of running useful software from the word go – surely of more importance than running very fast.

Shame About The PR

All and sundry have been singing the praises of the Electron. What won't receive much coverage is the media event that Acorn arranged to launch the machine to the technical press.

I won't bore you with explicit details but will just say that it was one of the worst such events that myself, and various other people with whom I talked, have ever attended.

About the only high point was the Electron cocktail that was served after the launch. For the record, to prepare an Electron cocktail take

1/10th bottle Champagne
1/6th gill Midori (melon liqueur)
1/3rd gill Brandy
1/3rd gill Cointreau

and stir well. Place melon balls in the bottom of glass and for best effect, sugar the rim of the glass.

Gary Evans

Short Form

Apologies for the rather truncated news section last month but the arrival of the Microdrive on the eve of our press day meant that a number of items had to be dropped at the last moment.

Things are thankfully back to normal this month.

Bye Bye Brainwave

As one of the magazines that took an active part in the preparations for Brainwave '83, it is with regret that we have to announce that the show has been cancelled.

The reasons for the show's cancellation are many and varied and the columns of *E&CM* are not the place to indulge in a post mortem of the exhibition that never was. We would however like to thank those at EMAP who worked enthusiastically on the Brainwave concept over the past few months and in particular to Richard Jansz. Richard has now moved onto the staff of *E&CM* and has thankfully recovered from his post-Brainwave blues.

Just as we were closing for press news of another aborted exhibition reached us. The Electronic Hobbies Fair which was to have been held at London's Alexandra Palace in late October was called off because of a general lack of interest.

Last year saw the first Hobbies Fair take place at almost the same time of year as the established Breadboard show. This meant that both shows suffered from a generally poor attendance both in terms of the paying public and exhibitions.

This year Breadboard has survived and will take place later in the year – *E&CM* will be at Breadboard and we will tell you more about the show and our involvement nearer the time.

Store Wars

With the competition between manufacturers combining with that between the various retail chains involved with the computer market the prices of computers are a' tumbling.

Atari 400s and 800s are now freely available for £100 and £200 respectively – the motive here is to clear stocks of these machines to make way for the soon to be launched 600XL and 800XL (to sell at £159 and £249 respectively).

Another price cut with stock clearing as one motive is the £45 ZX81 starter pack. This consists of a ZX81, a 16K RAM pack and a free piece of cassette software.

Sord's M5 has had £40 taken off its price. This makes the machine £149.95 – still rather expensive and Sord would do well to include the BASIC-G cartridge for this price, the M5 would then be very attractive.

Dragon are to sell their 64K machine for less than £200 and Commodore look set to slash the price of the 64 and to sell the VIC 20 at almost give away prices.

The Electron enters the fray at £199 but may well work its way down to the £150 level by early next year.

Whether the build-up to the Christmas period will start to firm up prices is yet to be seen but, for the moment, if you're on the look-out for bargains, London's Tottenham Court Road is the place to be.

States Snub Sinclair

The runaway success of Sinclair products in this country is unlikely to be repeated in the States. The problem is that the machines are just not as competitive in terms of price across the great divide.

For example the Timex TS1500 (Timex have the Stateside Sinclair license) costs $80. This is basically a ZX81 with extra memory and a Spectrum style keyboard. This undercuts the VIC-20 by only $10 and the monochrome, low-res display is hardly a match for even the long in the tooth VIC-20.

The US Spectrum lookalike is designated the TS2000 and this sells, in its 48K version, for $200 – exactly the same price as the Commodore 64.

Again the TS2000 is a very poor second in terms of performance – would you hesitate in buying a £125 64.

The major problem seems to have been the delay in translating Sinclair designs into their American equivalents. If the Spectrum had been available in the US at the start of last year there is every chance that it would have had a large sale if only for only a couple of months.

Circuit Update

Weather Satellite
July 1983

The RS crystal oscillator used in this project has been withdrawn. It's replacement does not provide a suitable voltage swing to drive the counter chain.

A circuit that provides a suitable replacement for the oscillator is shown right.

Dragon Cruncher

Many of you will be aware that both the Dragon 32 and the Tandy Colour Computer are built around the 6809 CPU. The structure of the machine is also very similar and it is therefore not surprising that programs written for one machine can be modified to run on the other.

Elkan Electronic's 'Dragon Cruncher' is a £7.95 cassette based, menu driven program that will convert Tandy Colour programs to run on the Dragon.

The company also distribute four US magazines that regularly publish software for the Tandy machine.

Elkan Electronics
11 Bury New Road
Prestwich
Manchester
M25 8JZ

See Us At PCW

Despite the fact that two shows have been cancelled (see elsewhere on these news pages) the end of September sees two shows that run almost concurrently.

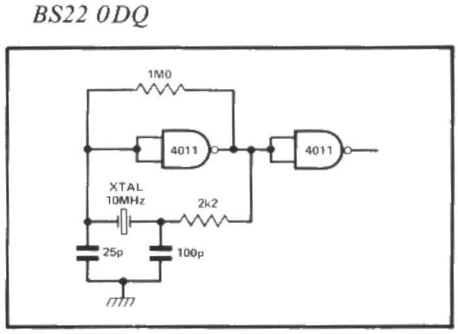
The Home Entertainment spectacular is a new show taking place at Olympia between the 17th and 25th of September.

A few days afterwards sees the start of the now well established PCW show at the Barbican. *E&CM* will be at PCW along with a number of our sister magazines. We hope you'll come along to PCW and if you do, please seek out the EMAP stand on which you'll find some friendly *E&CM* faces and a couple of exciting projects from recent issues.

Oric Monitor

New from Kenema is a powerful Oric monitor. The £15 software package provides facilities that include Memory Display and modification, some powerful program debugging aids and a disassembly display.

Kenema
1 Marlborough Drive
Worle
Avon
BS22 0DQ



COMPUTER PRODUCTS

U Know Who

U Microcomputers has recently introduced an extremely versatile expansion system for the Spectrum.

The initial products in the range are a four slot buffered backplane, a three slot extension backplane – together allowing seven boards to be connected to the Spectrum, a prototyping board and two interfaces.

The first interface, designated the USP – 232D, is a powerful dual channel serial interface designed around the Z80 DART IC. The board provides an LPRINT and LLIST patch.

The second board is a parallel interface built around the Z80 PIO. 16 input or output lines are provided in addition to four control lines. In addition a low cost add-on kit (USP – CENT) provides a cable and LPRINT and LLIST drivers.

Both the interface boards are supported by comprehensive manuals. Further cards are at present under development . . . watch this space.

The expansion system is as yet only available via mail order – prices and additional information from:

U Microcomputers
Winstanley Industrial Estate
Long Lane
Warrington
Cheshire
WA2 8PR

Monitoring Oversights

In our monitor of last month we overlooked the activities of Citadel Products who produce a monitor that is both designed in Britain and uses components that do not originate in the Far East.

With customers that include British Telecom, the MoD and Phillips, the 101 monitor obviously has something going for it. The model is a 22MHz unit with a green P31 phosphor. The 101 costs £89.50 plus VAT and in addition, a 12V version of the unit (RL102) is available at £99.00 plus VAT.

The company also manufacture a comprehensive range of keyboards.

Further details from:

Citadel Products
50 High Street
Edgeware
Middlesex
HA8 7EP

Minor Miracles

The Electronics and Computing Monthly RGB interface for the Spectrum has been delayed because of technical problems but news of a new commercial interface with the same aims should make up for any disappointment caused by the delay in our design.

The MI3 interface comes from Miracle Systems, a company formed by ex-Sinclair Research Engineer Stuart Honeyball.

The MI3 is a 'black box' which plugs into the back of the ZX Spectrum and has two BBC Micro compatible 6 pin DIN sockets to provide both TTL and IV p-p linear RGB signals suitable for driving most colour monitors. An MI3 with an RGB monitor gives the user an extremely high quality display since four adjustable components in the ZX Spectrum and the PAL system are completely bypassed – the MI3 even has its own internal 8K CMOS static video RAM. Shimmer and colour fade problems are eliminated.

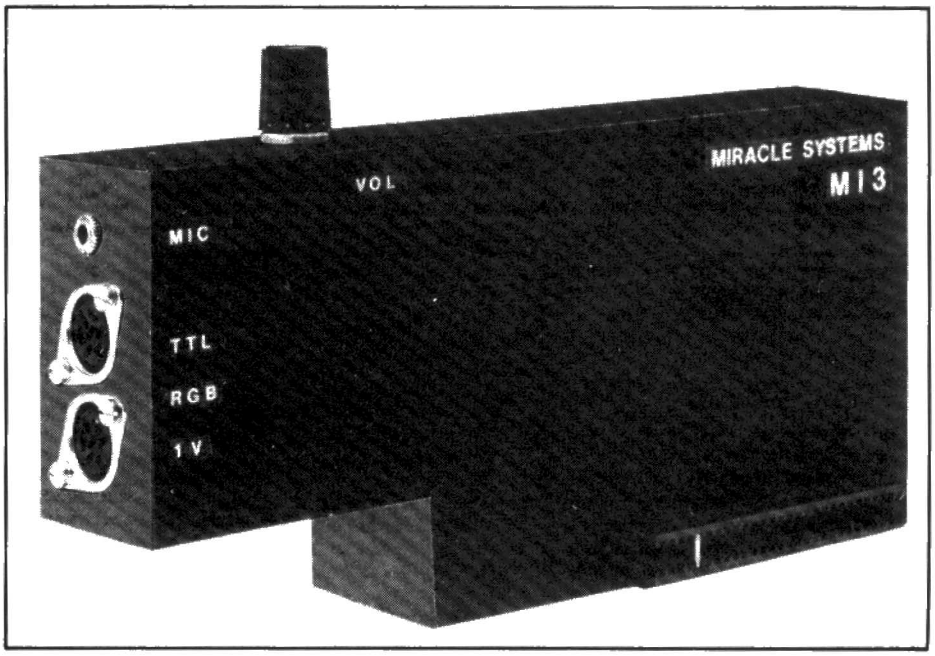
Games users will be pleased to hear that the MI3 also amplifies the ZX Spectrum sound using a 2.5 speaker and volume control.

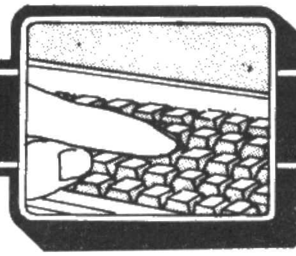
A 3.5 jack socket located on the MI3 is used in place of the ZX Spectrum's MIC socket so that programmers who frequently use the LOAD and SAVE commands no longer need to insert and remove the EAR jack between these operations.

The MI3 will work with all ZX Spectrum software and does not require any special software to drive it. The ZX Spectrum expansion port is duplicated so that other peripherals may be connected.

Further information about the MI3 may be obtained from:

Miracle Systems Ltd.
6 Armitage Way
Kings Hedge
Cambridge
CB4 2UE





Speech Storage System

The limitations of the various forms of speech synthesis systems are mentioned in the 'Sweet Talker' review elsewhere in this issue. A system that overcomes the majority of the shortcomings of such systems is the Voxbox.

This device is capable of storing speech on disc and the reproduced speech retains most of the intonation or stress or the original.

The Voxbox comes complete with full operating software that allows single words, phrases or sentences to be easily stored and manipulated.

The method of digitisation is 'liner delta modulation' (see next month's instalment in our 'optimal coding' feature for an explanation of this technique). There is a trade-off between amplitude and frequency with LDM which means that a choice between either storing high frequencies or large amplitudes must be made. The digitisation rate can be software selected to lie within the range 24-8K bits/second.

The Voxbox is available for £80 as a stand alone unit or for £89 in include software, a speaker and a microphone.

Further details are available from:

Multiplex Computer Services
250 Eastern Road
Brighton
Sussex
BN2 5TA

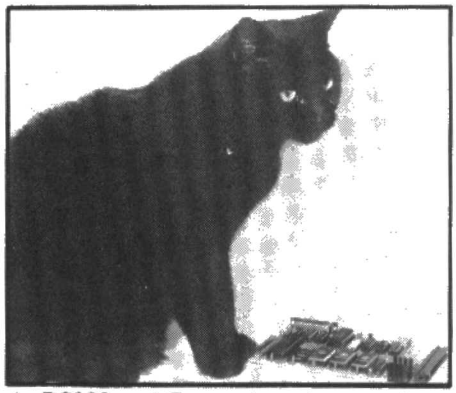
Spectrum Goes Forth

Taking a leaf from the Sony advertising campaign, David Husband has included a Black Cat in one of his views of his latest product – a Spectrum Forth I/O cartridge.

The unit features a 12K implementation of Fig-FORTH, a full RS232 interface, via an 8251, and a total of 24 bits of parallel I/O via an 8255.

In use the Spectrum's BASIC ROM id switched our and replace by the FORTH interpreter. In addition to providing the FORTH language to cartridge endows the Spectrum with a number of other features that include:-

★ A multi-tasking FORTH operating system allowing separate Background and Foreground programs to execute simultaneously and transparently of each other.
★ A Z80 Assembler for machine code FORTH definitions.
★ A built-in terminal routine to support a modem and allow access to bulletin-board systems. This routine can be used from FORTH to BASIC.
★ A crystal-controlled baud rate generator for RS232, giving a choice of 8 different baud rates between 765 and 9600 bauds.
★ A Machine Code Monitor which can operate in any number base, carry out simple one-line assembly and disassemble.

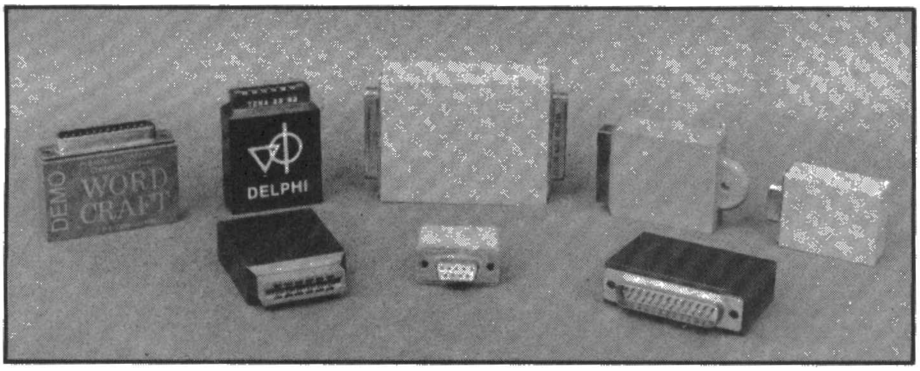


★ RS232 and Centronics printer routines, usable from FORTH or BASIC RAM Disc Simulation and support of Micro-drives when they appear.
★ Full use of the colour, sound and graphics facilities from FORTH.
★ Full use of the Cassette routines from FORTH.
★ 4K of ROM space is available for future software enhancements, such as floating-point or LOGO.

In order to fully appreciate the facilities of the cartridge a book 'Advanced FORTH', which describes all the Cartridge's features, is available from Sigma Technical Press – the book is included in the purchase price of the cartridge but costs £7.95 as a separate item.

The cartridge is available both from Watford Electronics and from the designer David Husband at:

2 Gorleston Road
Brankstone
Poole
BH12 1NW



Day Of The Dongle

Wordcraft Systems have developed a range of software protection devices that have won the rather quaint name of 'dongle' for themselves.

Each dongle contains a unique code which is interrogated during the operation of the program – if a Dongle is not present, or if the Dongle's code differs from that embedded in the software, the program will not run.

This simple idea does not prevent users from making back up copies but, as you need a Dongle at run-time, it does confine software to a single machine.

Dongles range in price from £2.50 for a VIC 20 variant to £15.00 for an IBM PC version.

Minimum order quantity is 100 and any software authors may well consider giving a dongle with their product – the increased revenue resulting from the elimination of pirate copies should easily pay for the hardware.

Wordcraft Systems
43 Farley Road
Derby
DE3 6BW

Eprom 81

The ZP-4000 is a sophisticated EPROM Programmer designed for use with the ZX81. The unit is fitted between the ZX81 and a 16K RAM pack and features an on board 25V generator, PIO, ZIF socket and a 2K operating system.

The ZP-4000 allows a variety of operations to be selected from its menu display. These include copying PROM to RAM, displaying or editing the RAM image, examining PROM content and a simple transfer to and from BASIC.

The ZP-4000 will handle either 2516 or 2532 of inbuilt safety checks.

The ZP-4000 is available from:

Enterprise Technology
PO Box 140
Wigan
WN3 6LF

at a cost of £63.25 all inclusive.

THE ELECTRON

For £199 the new Acorn Electron offers many of the facilities of the BBC Micro. The questions are what if anything has been sacrificed to produce a lower cost machine and what has the Electron got that the competition hasn't. Mike James reports.

It is perhaps unfair simply to compare the Electron to the BBC Micro and find it wanting. For example, it is all too easy to concentrate on the fact that the Electron is slower than the BBC Micro and miss that point that the Electron is still up with the front runners when it comes to spped! In the same way that Electron may not have three sound cahnnels and a complex ENVELOPE command but its sound capabilities are still very good when compared with other machines in the same price bracket. It's a sure bet that in the future we will hear a great deal of criticism of the Electron based on a comparison with its more powerful stable mate but this is to overlook two of its very remarkable features – its small, compact size and its small, affordable price tag. The Electron is a high performance machine in its own right and it is this fact that should never be overlooked in the course of all the inevitable comparisons.

The Electron is the third personal computer to be produced by Acorn. The first was the well known Acorn Atom, the second was the even better known BBC Micro and now we have the Electron. The only sign of uneven progress in Acorn's machines are their names. In the case of the Atom and the Electron Atom displayed a clear cut preference for physics in a way that other manufacturers prefer fruit! The reason why Acorn didn't name their second machine Positron, Neutreno or Higgs Boson is well known to everyone. The BBC part of the BBC Micro's name certainly brought an excellent machine to the attention of the public rather more quickly than just good reviews in computer magazines and working to gain the approval of an outside organisation rather then to just produce what was technically easy must have added to the quality and overall usability of the machine. However, the price that Acorn have had to pay for these advantages is high.

Readers of *E&CM* might be clear that Acorn make the BBC Micro but my guess is that much of the new Electron's potential market will wonder who Acorn are! Of course Acorn will be able to make reference to the success of the BBC micro in their advertising but my guess is that they will have to work hard to promote it.

Outline Of An Electron

The Electron is a physically small machine. However, to say that it is 13 inches by six and a half inches by one and a half inches fails completely to convey the good looks and 'chunky' feel of the machine. The typewriter-style keyboard is made of the same cream plastic as the rest of the case. Although there are fewer keys than on the BBC micro the Electron has some surprises in store. On the front face of every key, and surprisingly visible in brown lettering, is a BASIC keyword, making it possible to enter BASIC programs using single keystrokes in the style of the Spectrum. The top row of keys double as programmable function keys – a very useful facility.

Setting up the Electron is simplicity itself. All the necessary connectors – for a UHF modulator, video signal, RGB monitor and cassette are to be found neatly lined up on the left hand side of the case. A 'raw' power input socket is positioned on the righthand side ▶

and accepts the plug from the separate transformer. The practice of building the power transformer as a separate unit is almost the standard for low cost micros and the Electron is no exception with a built in 13 Amp plug in a rather large cream unit. Underneath the Electron's case is a recessed PCB connector intended for expansion modules. The fact that there are two moulded screw threads on either side of the connector (obviously intended to be used to secure such modules) is a clear indication of how well thought out the Electron's design is.

In use the Electron is very pleasing. The keyboard feels good and the video display is clear and steady. The small size of the Electron is a bonus in that it leaves so much space on the desk top for tape recorders and books while you are working with it.

Hardware

The Electron's hardware is superficially very similar to the BBC Micro's. Its 32K of RAM, 32K of ROM, double speed 6502, graphics modes and sound generator give is a specification that matches the BBC Micro in all but the fine detail. However, the Electron is a new machine in that most of these features are implemented using quite different electronics from the BBC Micro. In particular, the role of the video 6845 video generator has been taken over by a single custom ULA chip. In fact you could say that the Electron is built around a double speed 6502 and a very large ULA which handles the cassette, the entire video generator circuitry including the palette RAM and serialiser and the sound effects generator. Thus the Electron's large ULA does the job of two ULAs, the 6845 video generator and the sound generator in the BBC Micro and neither of these familiar components are present. This is clearly an indication of how fast technology, and ULAs in particular, are improving. No doubt next year we will see a complete computer including CPU and RAM on a single ULA, but predictions are always dangerous!

The use of a ULA to perform so many functions is obviously one of the main reasons why the Electron is so small. However, it is mimportant to realise that the Electron is so much smaller than the BBC Micro because it contains none of the on board expansion options. That is, the Electron cannot be expanded to use disks, a printer, joysticks or an econet interface without the use of external modules. Although it is easy to see why Acorn didn't include disks or an Econet interface – these items are expensive and easy to do without – it is more difficult to see why printer and joystick interfaces have been left out. The long term success of the Electron will certainly be influenced by the eventual range and cost of the extra modules necessary to extend its capabilities to more serious and general applications.

The Electron's graphics modes are the same as those on the BBC Micro with the notable omission of mode 7, Teletext graphics, see **Table 1.**

Any BBC programs that use mode 7 will automatically be run in mode 6 on the Electron. This makes sense because mode 6 gives the same 40 characters by 25 lines of mode 7 but it uses 7K more of memory so you might find that some programs run out of space. Another problem with using mode 6 to replace mode 7 is that none of the special teletext features such as double height characters are supported and teletext graphics just appear jumbled on the screen. No doubt someone will provide a program to make mode 6 look more like teletext and then the only problem will be the shortage of memory. It is clear why mode 7 was left out of the Electron, a teletext character generator is not a cheap component, but this leaves the Electron without a text mode that is economical on memory. Apart from this omission, the Electron offers a wide range of graphics options, from a high resolution two colour mode to a medium resolution 16 colour mode, plus two text each only modes offering 25 lines with either 40 or 80 characters to a line. (For a full list of modes see **Table 1**). Although the Electron generates its graphics using a different method from the BBC Micro, from the user's point of view it is virtually indistinguishable. You can mix text and graphics freely anywhere on the screen and can define your own graphics characters. Any BBC Micro program written entirely in BASIC should run on the Electron without any modifications. But any program that tries to make use of the non-existent 6845's control registers is likely to cause some trouble.

The Electron's sound generator provides only one tone and one noise channel as opposed to the BBC Micro's three tone and one noise channel. However, the same BASIC sound commands are available including a slightly restricted form of the ENVELOPE command. The SOUND command produces a noise without causing BASIC to pause making synchronised sound and movement easy. Using ENVELOPE you can only control the way the pitch varies but only within a preset amplitude pattern. As a result BBC Micro programs will run on the Electron but they don't always sound as they were intended to! However, you can still produce simple tunes and sound effects and the restrictions on the ENVELOPE command might make it easier to use for a beginner!

The Electron is slower than the BBC Micro by a factor that varies from 1.5 to 3 depending on what it is doing. The reason for this speed loss is that its 32K of RAM is organised as four 64K bit chips. Thus it takes two memory accesses to handle a single byte. Of course this extra complication is kept hidden from the user by the ULA but it does slow things down. This reduction in speed is very noticeable when using graphics. However, before leaping to the conclusion that this loss of speed is a major drawback, it is important to realise that while the Electron is slower than the BBC Micro it is far from being a slow machine. By the standard of other micros the Electron is *fast!* On standard benchmarks it comes out about as fast as the Sirius 1 and the APPLE III and is way ahead of its similarly priced competitors.

The Electron's tape system is the same as the BBC Micro's but only works at 1200 baud. In practice this restriction is unlikely to cause any problems because with a properly aligned cassette recorder 1200 baud is very reliable. However, it is worth bearing in mind that the Electron cannot load 300 baud BBC Micro tapes.

The Electron has no spare ROM sockets so software contained in plug-in ROMs cannot be used without an external module. However, if you are prepared to remove the BASIC ROM then other ROM software that doesn't need teletext graphics or any of the other missing facilities can be substituted. So, for example, Acornsoft's word processor, View, could be used.

The Software

The Electron's BASIC is officially called Acorn BASIC but it is in fact virtually identical to BBC BASIC. It includes all of the advanced features that have made BBC BASIC so popular – procedures, local variables, indirection operators etc. This coupled with the fact that it is interfaced to the Electron's hardware through 16Ks worth of Machine Operating System (MOS) that is very similar to the BBC Micro's MOS makes the Electron's software a BBC Micro look alike. The PLOT command has one extra facility in that it will fill a screen line with colour, this opens up the possibility of Electron programs not working on the BBC micro. Most of the appropriate *FX (operating system) calls are present and the VDU drivers accept the same control codes as in the BBC Micro.

TABLE 1

Mode	No. of Characters	No. of Pixels	No. of Colours	Memory Used
0	80 x 32	640 x 256	2	20K
1	40 x 32	320 x 256	4	20K
2	20 x 32	160 x 256	16	20K
3	80 x 25	(text only)	2	16K
4	40 x 32	320 x 256	2	10K
5	20 x 32	160 x 256	4	10K
6	40 x 25	(text only)	2	8K

A welcome feature of the Electron is the presence of a 6502 assembler in ROM. This can be used to mix BASIC and assembler in

a way that is particularly suited to the beginner. The only difference between the Electron assembler and the BBC Micro's assembler is the provision of some new pseudo ops that can be used to store constant data in memory. This is a useful improvement but again raises the possibility that Electron programs might not be compatible with the BBC Micro so such features have to be used with care if developing programs for both machines is of practical concern to you.

Documentation

The Electron comes with a manual that is heavily based on the BBC Micro's manual and so it is a weighty tome suitable for holding doors open! It gives a good introduction and summary of Acorn BASIC, the MOS and assembler. The section on assembler has been much extended and now shifts the manual's balance away from BASIC. For this reason among others Acorn are supplying "Start Programming with the Electron" by Masoud Yazdani with every computer. This is a short introduction to BASIC for the complete beginner but it does cover rather too much ground for its 117 pages. Rather more interesting is the introductory cassette that comes free with the Electron. This includes quite a few games and demonstrations that can be listed and examined to find out how to do some quite advanced things.

Electron vs BBC Micro

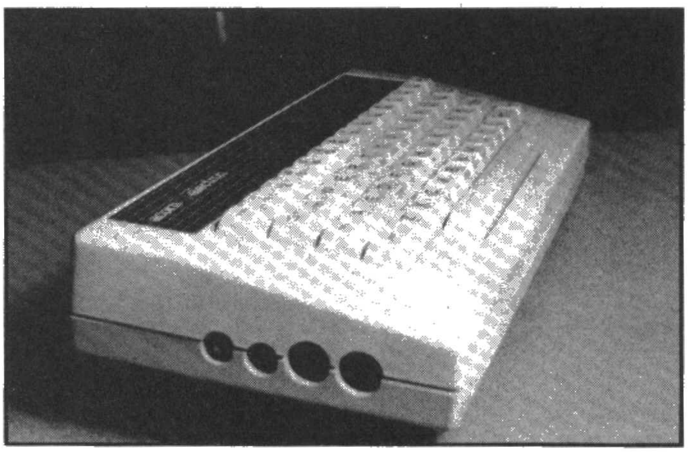
It has been impossible to avoid continually making comparisons between the Electron and the BBC Micro because the Electron is best summed up as a 'stripped down' BBC Micro which runs slower. It is stripped down in the sense that it has all of the features of the BBC micro apart from teletext graphics mode, a second cassette speed, space for extra ROMs and the full range of interfaces. Specifically, it lacks a serial interface, joystick interface, 1 MHz and Tube interfaces and the printer and user ports. It will be possible to add all of the missing features apart from teletext graphics and 300 baud cassette loading. However, the question really is why would anyone prefer to buy the Electron rather than a BBC Micro?

The most obvious answer is price. The Electron provides a low cost route into the world of 'BBC computing'. In other words, with the Electron you can take advantage of all the software, the books and the media razamatazz that has been attendant on the BBC's Computer Literacy Project. Although you will have the possibility of up-grading to a fuller system as finances allow the Electron is a more than adequate computer as it stands. It has a specification that makes it suitable for use as a games computer, an educational computer, a graphics computer, a low cost development system... And by adding only one or two of the promised expansion modules its range is likely to be even greater. The Electron doesn't entirely fill the role currently occupied by the BBC Micro but there are many applications that the Electron can be used in at lower cost and without a noticeable loss of performance.

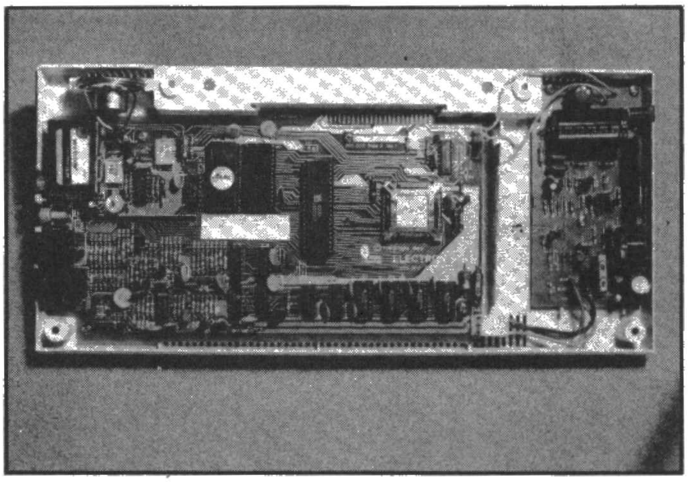
Software Compatibility

An important feature of the Electron is ist ability to run programs designed for the BBC Micro. To try to discover just how compatible it really is a few BBC Micro programs were tried on it. The first surprise was that all of the programs contained in "21 Games for the BBC Micro" loaded and ran immediately. The reason why this was a surprise is that some of the programs are very large and use Mode 7 for output. Although they all worked without error, the slower speed of the Electron made some of the games unplayable. However, as all of the programs contained delay loops to slow them down in order to run on the BBC Micro, correcting this problem was easy. A more worrying problem was the way that the sound effect came out differently. A rifle shot in one program sounded more like a car horn! This would obviously need more work to correct.

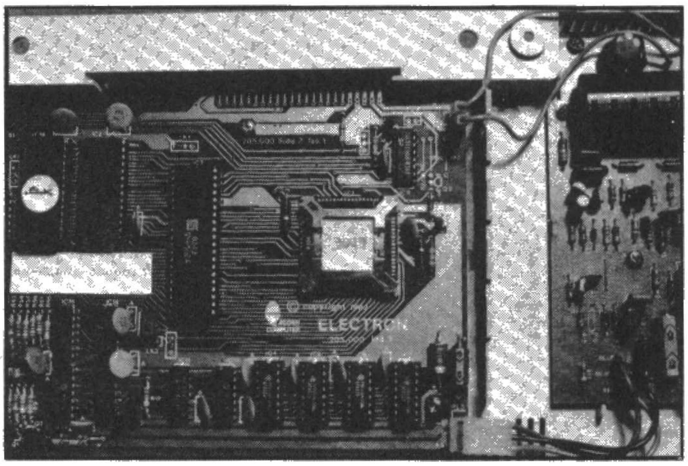
After trying this wide range of BASIC programs the next thing to try were some large Acornsoft programs. Some of these worked after a little coaxing. The main trouble was that in each case the initial loader program used mode 7 teletext graphics and this just came out as a random jumble. However, once the main program was loaded and running most of the programs put up a good attempt at working



The four sockets at the right hand edge of the case provide UHF and composite video outputs as well as the RGB and cassette I/O connectors.



The Electron features the same switch mode PSU as the BBC Micro (board at right of case). The main PCB shows a dramatically reduced chip count compared to the BBC.



Moving in for a closer view of the Electron's ULA (centre). This incorporates the video and sound controllers as well as the cassette I/O and various control and decode functions.

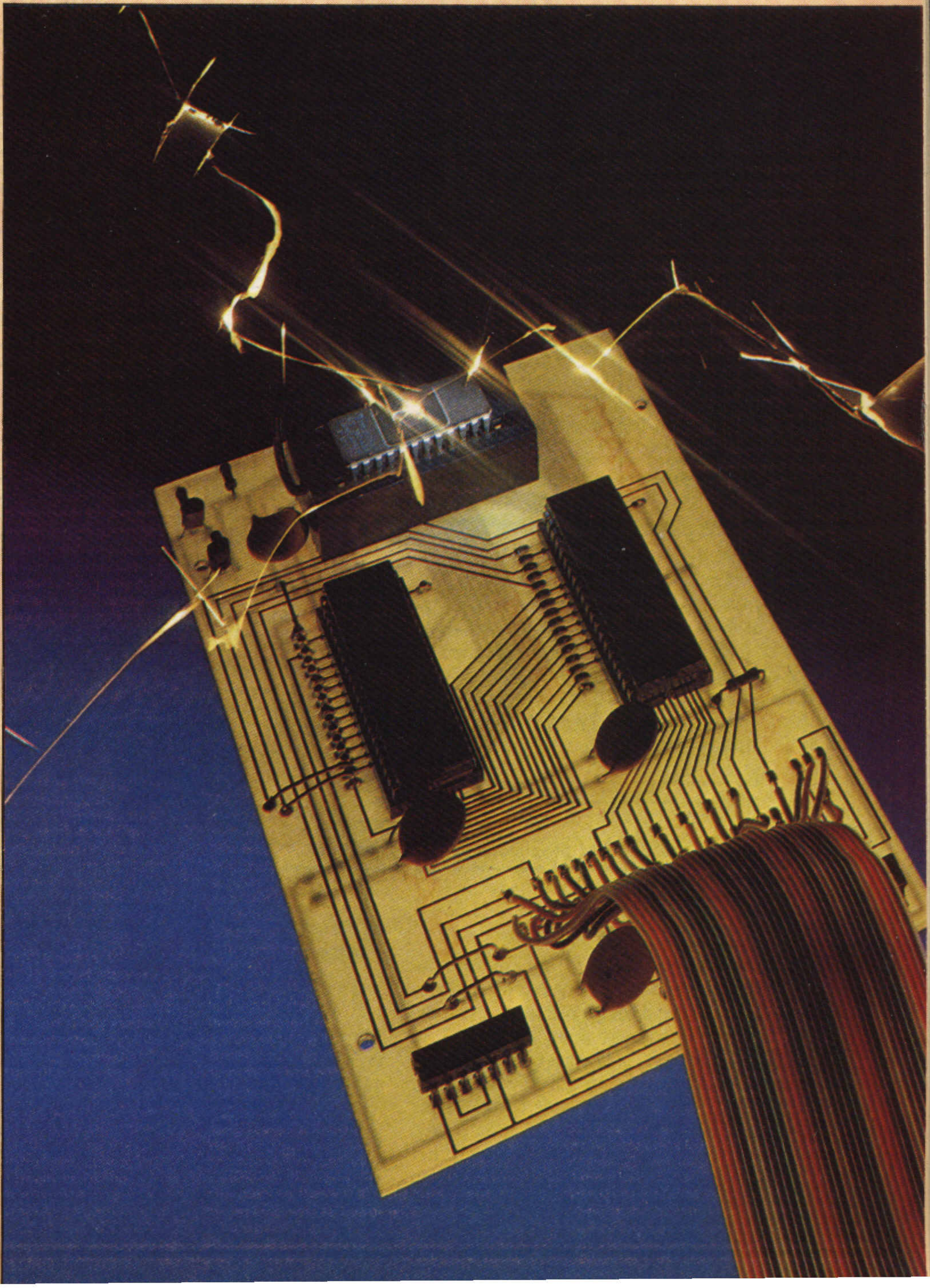
apart from the decrease in speed noticed earlier. The exception to this was PLANETOID which relies on using the 6845 to produce horizontal scrolling. Without the 6845 the result was unusable!

The conclusion to all this is that the Electron really does behave as predicted. A program that makes no direct use of the BBC Micro's hardware will run on the Electron – it may sound a little different and work slower but it will run after a fashion. This should mean that most of the software companies including Acornsoft should be able to offer Electron versions or even dual BBC/Electron versions of their programs very quickly.

E&CM

EPROM PROGRAMMER FOR THE BBC MICRO

Peter Simpson and Brian Alderwick describe the hardware and software details of an EPROM 'blower' that can program EPROMs for use with the BBC's sideways ROM sockets. Now it's possible to 'blow' your own EPROMs with, perhaps, a machine code disassembler – the limit is set only by your imagination and 16K of memory.



Most users of microcomputers must surely at one time or another have wished for useful routines to be immediately available without having to load from disc or, even worse, having to search through never ending tapes. A facility which avoids these problems, is available on the BBC micro in the form of the sideways ROM sockets. This article describes the necessary hardware and gives a complete software listing for an EPROM "blower" capable of programming EPROMs to fit into these sockets.

Given the facility to "blow" your own EPROMs it is then up to you to write the programs to go into the EPROMs. Perhaps a machine code disassembler or a utility to list BASIC variables and their contents take your fancy – the limit is only set by your imagination and 16K of memory.

An EPROM (Erasable Programmable Read Only Memory) is an integrated circuit which can be programmed by applying certain electrical signals to it. Once programmed, the memory contents can be considered semi-permanent and are retained even when the computer is switched off. If it is required to re-program the EPROM, it must be exposed to "hard" ultra violet light for approximately twenty minutes through the clear window on the top of the chip, after which it is ready for re-use. Hence the EPROM is a very versatile chip not only for one off home use, but also for commercial purposes when the total numbers do not warrant the expense of permanent ROM manufacture.

Because of the restriction in the amount of memory addressable by the 6502 microprocessor, Acorn have provided a way of switching any one of four (potentially sixteen with hardware expansion) alternative banks of 16K into the same part of the memory map. This area is known as the paged ROM area and is located from &8000 to &BFFF (see page 500 of the user guide). BASIC resides in this area as do options like Wordwise, View and the Disk Operating System. Thus the normal limitation of 64K of memory can become 108K in the standard micro and ultimately 304K – Acorn have managed to get the proverbial

quart into a pint pot! The machine operating system (MOS) needs to be a series one version, that is 1.0 or later since the 0.1 version does not support the paged ROM system. Series one operating systems are now available from ACORN dealers.

The four paged ROMs are located in 28 pin dual in line sockets (IC52, IC88, IC100 and IC101) at the front right hand side of the main board. The fifth socket in from the right (IC51) contains the machine operating system ROM and should not be touched. One of the paged ROM sockets will already be filled with BASIC and if you have disc drives, another will be filled with the disc operating system. It is interesting to note that if you have a series one MOS, the computer will on power up, select the language ROM which is located nearest to the right.

There are many kinds of EPROMs on the market and this can lead to confusion. The BBC micro will accept 2764 (8K byte) and 27128 (16K byte) EPROMs and it was decided for reasons of simplicity, to design a programmer limited to these two types. The speed rating of the EPROMs should be 300 ns, although the authors have programmed many Hitachi and Intel 450 ns versions and all have worked perfectly. Only these two makes have been tried, therefore they are only types presently recommended for use in this programmer.

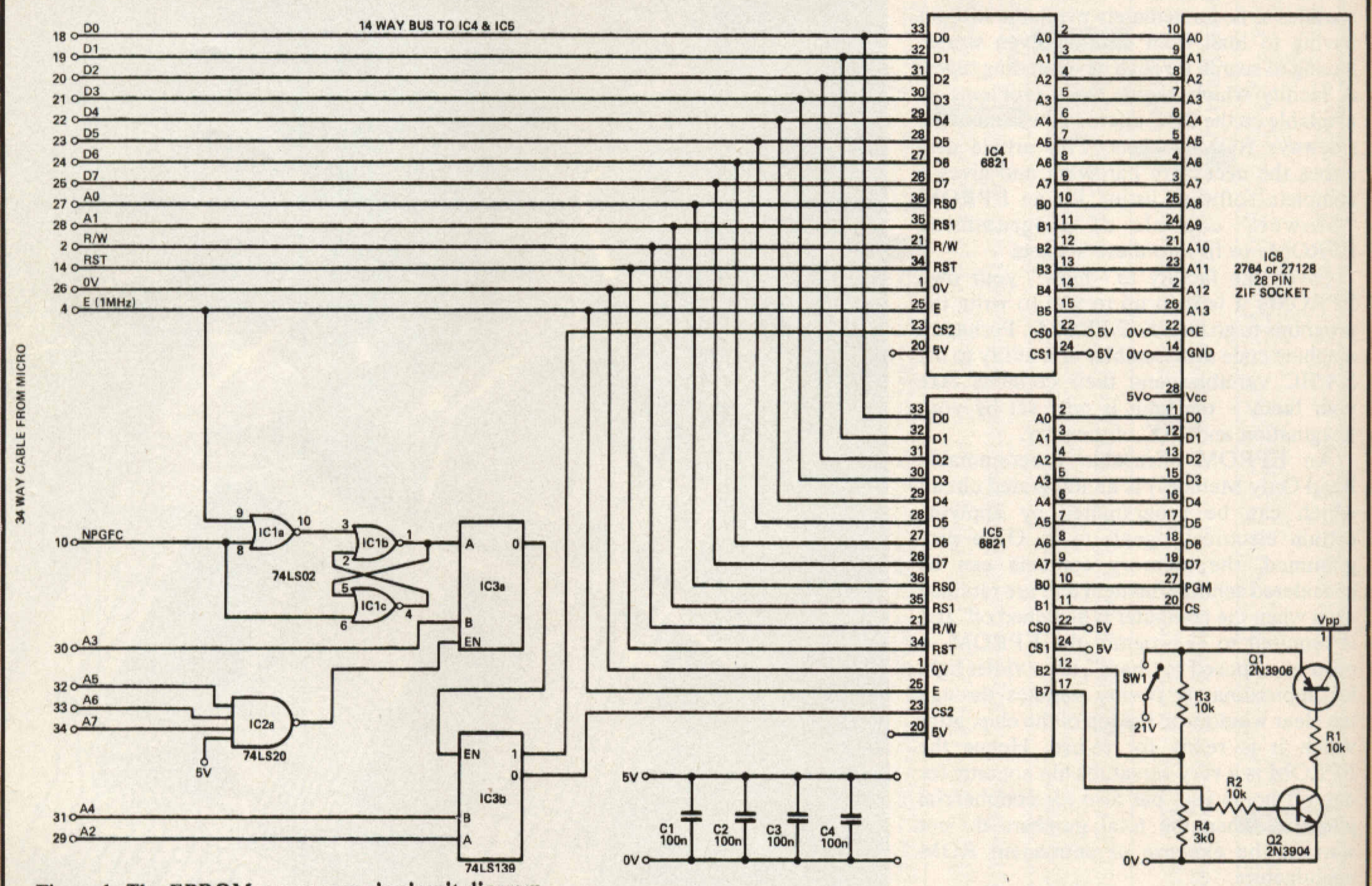


Figure 1. The EPROM programmer's circuit diagram.

Circuit Details

A complete circuit diagram of the programmer is shown in **Fig 1.** The design is attached to the 1MHz bus and uses the NPGFC and address lines A0 to A7 to decode memory addresses &FCE0 to &FCE3 and &FCF0 to &FCF3 inclusive. These are in the area set aside by Acorn for user applications in the FRED area of this interface. The value of this is that other hardware can be simultaneously connected to the bus, such as the Teletext unit and IEEE 488 interface, without causing interference to one another.

The address decoding is carried out by IC1 to 3. IC1 is a quad two input NOR gate and three of these gates are used to "clean up" the NPGFC pulse. IC2 is a four input NAND gate and IC3 a dual two to four line decoder. These decode the above addresses and generate the chip select and register select functions of IC4 and IC5, which are Motorola 6821 Peripheral Interface Adaptors (PIAs).

Ports A and B of IC4 generate the fourteen address lines, whilst IC5 port A generates the data lines and port B various control lines. IC6 is the EPROM to be programmed and a zero insertion force socket is used to hold this chip. Transistors Q1 and Q2 form a switch for the 21 volt programming voltage.

This supply also has a manual switch which acts as a safety measure. The position of the switch is sensed by port B2 on PIA IC5 **(Fig 1)** and positional change is prompted by the software.

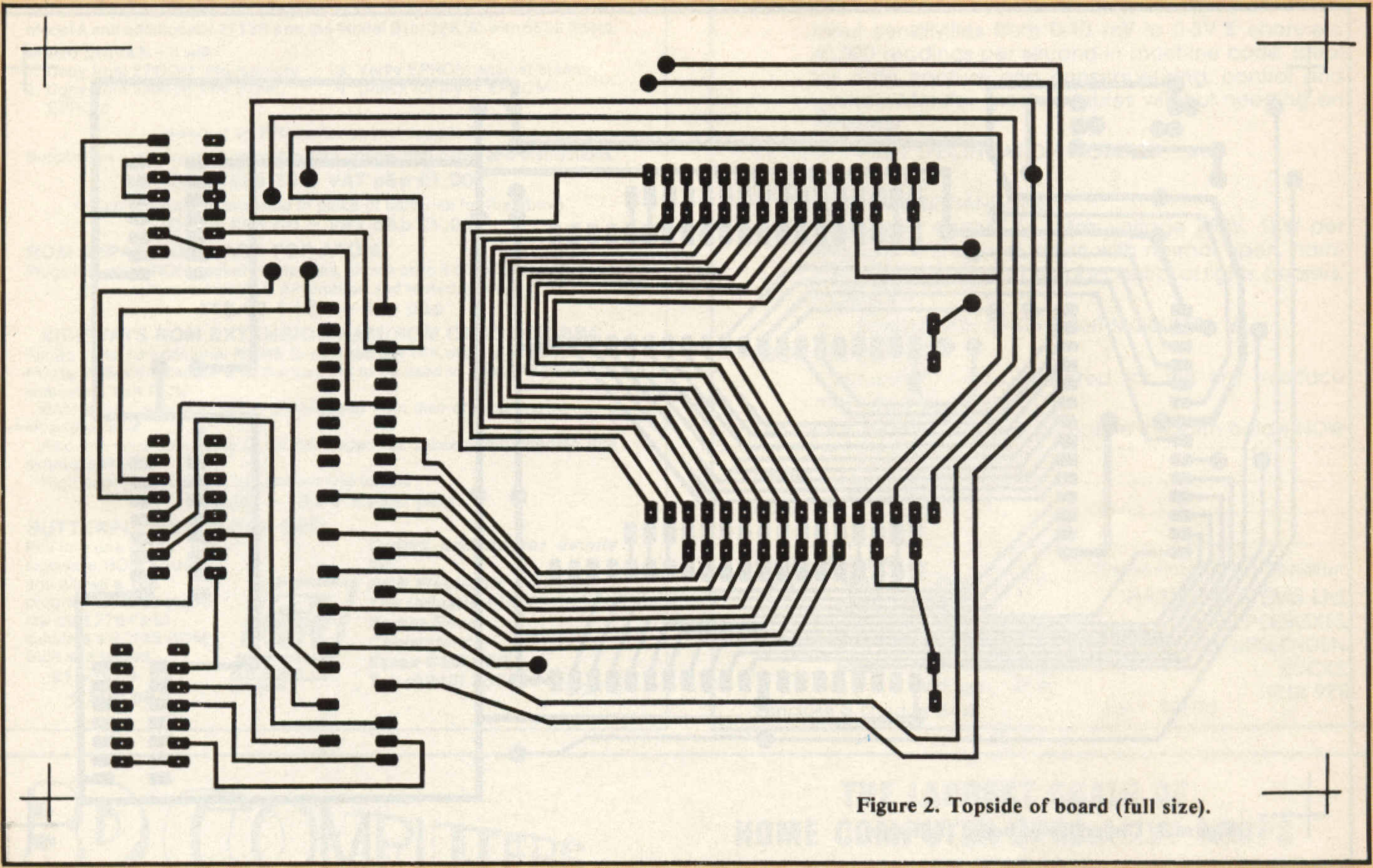


Figure 2. Topside of board (full size).

Construction

All the ICs are mounted on a double sided printed circuit board which has been designed with home construction in mind, i.e. through links have been provided instead of the commercial practice of using plated through holes.

The layout of the printed circuit board is shown in **Fig 2** and **3.** When the board is printed the two layouts must be in close register since there is little room for error.

It is important to note that, when drilling the board, all the 34 way cable terminations and, with three exceptions, all the bridge connections, are pads and should not be drilled. These exceptions are at the IC end of the three bridges 'north' of IC4. The bridges can be made from any stiff wire, e.g. scrap resistor ends, but it is important to make sure that the bridges do not make contact with any of the tracks that they are supposed to be bridging!

The components are fitted to the board as shown in **Fig 4** and the photograph clearly shows the layout of the finished board. Note that IC1 pin 1 faces 'north' whilst IC2 and 3 face 'south'. Construction is straightforward but is made easier by the use of a small point soldering iron.

The 34 way insulation displacement cable (IDC) is soldered to the pads on the PCB and the other end is fitted with an IDC 34 way header plug to fit the BBC micro. Make sure pad 1 goes to pin 1 on the plug. The cable should be at least 800mm long.

The constructor is left to make a suitable power supply (5 volts at 300 mA and 21 volts at 60 mA) and provide a housing. With care the zero insertion force socket can be made to fit flush with the top of the case, producing a neat finished product.

Programming Requirements

Before the software is described it would be a good idea to describe the EPROM programming requirements. Firstly the programming voltage, Vpp (21 volts), should be applied to pin 1 and the chip select line held low. Then to program each byte the address and data lines must be steady and a pulse of 50 ms duration (plus or minus 5 ms) at logic 0 should be applied to the PGM pin 27. Programming a 16K EPROM at 50 ms. per byte takes about 14 minutes.

Software Synopsis

The software makes extensive use of a RAM buffer which physically resides at &3C00 to &7BFF although the software expects addresses in the range 0 to &3FFF for compatibility with the EPROM addresses. The buffer acts as a temporary store to which the contents of the EPROM can be copied or from which the EPROM can be programmed. A variety of options for manipulating and displaying the contents of the buffer are provided in the software.

The software has been written to be fully compatible with disc or cassette, the program senses which filing system is in use. Users of disc should choose start up option three which is selected by the command *OPT 4 3. This will *EXEC the short file !BOOT which can be a simple one line file created with *BUILD consisting of:-

```
1 CHAIN "EPROM"
```

Thus a SHIFT/BREAK will autostart the programmer. Cassette users should employ CHAIN "EPROM". The software is split into two parts. The first program is in BASIC and assembles machine-code from &3A00 to &3B98 and it is this which actually ▶

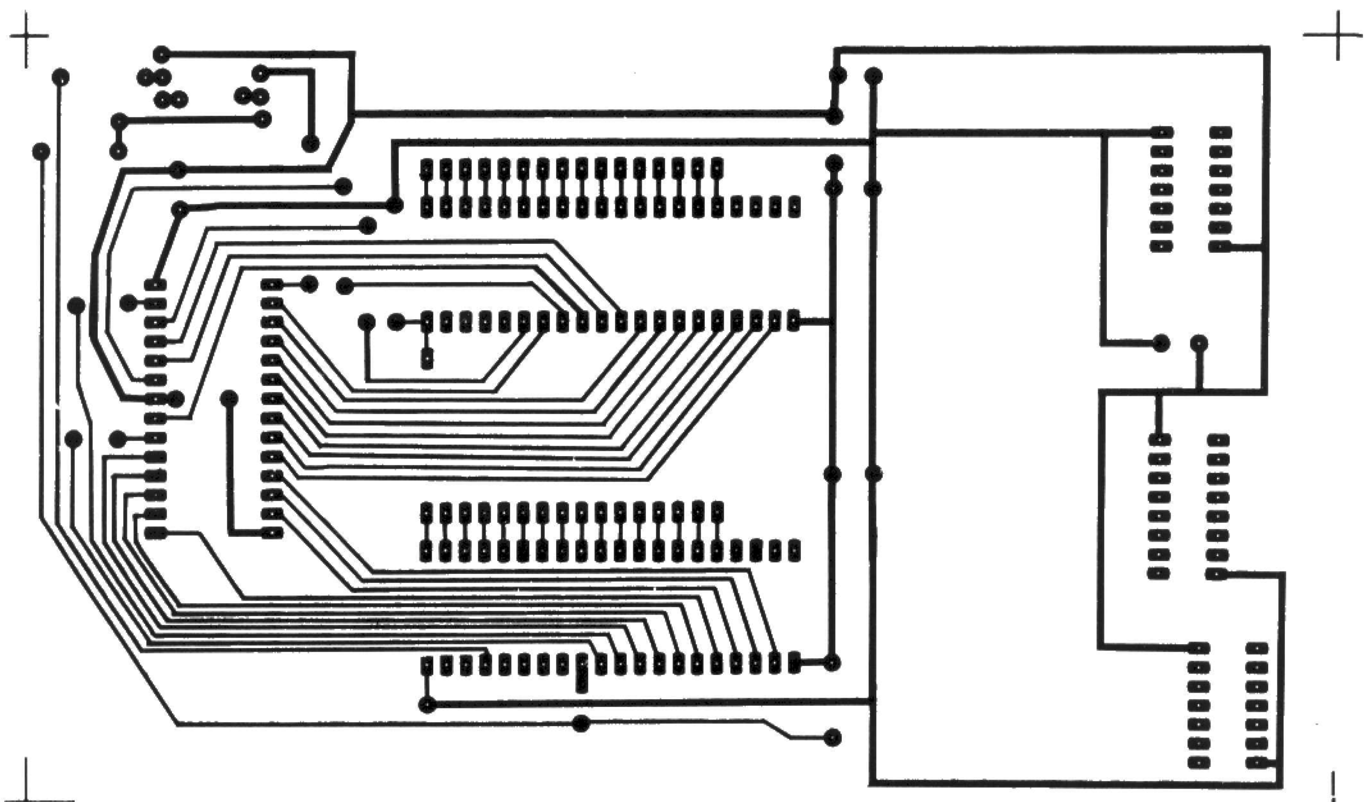


Figure 3. Underside of board (full size).

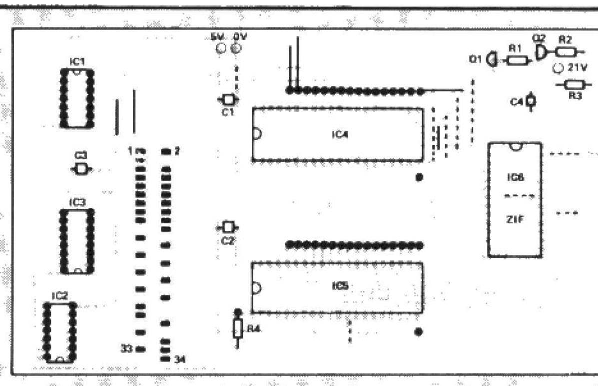


Figure 4 (left). The EPROM programmer's overlay showing the positioning of components.

● = THROUGH PINS (SOLDER BOTH SIDES) (IC1-3 SOLDER LEGS BOTH SIDES)
▪ = SOLDER 34 WAY CABLE TO PADS ON TOP OF BOARD
○ = POWER SUPPLY CONNECTIONS
— = WIRE BRIDGES ON TOPSIDE PADS
--- = WIRE BRIDGES ON UNDERSIDE PADS

performs the hard work of reading writing and verifying the EPROM. The second program is also in BASIC and offers a menu of options related to programming the EPROM, at times calling up the machine code routines assembled by the first program. The second program contains many REMs and is too large to fit into the RAM available in a disk based system. However, if line 205 PAGE=&1100, is added to program "EPROM" this resets PAGE, reclaiming some of the disk workspace allowing the programs to run as written. However, this is not recommended and it is suggested that REMs and blank lines are left out. If this is done the programs will run in the normal way with page set to &1900.

For ease of use a menu driven format was adopted. The options provided are shown in **Table 1.**

When first used the program automatically goes to option 1 to select the EPROM type since this must be provided to allow other options to work.

Each of the options, except number ten, gives the user further options to return to the main menu or to carry on with the one selected. In options 4, 5 and 6 the program uses the buffer from address 0 to the length of the EPROM chosen by option 1.

TABLE 1

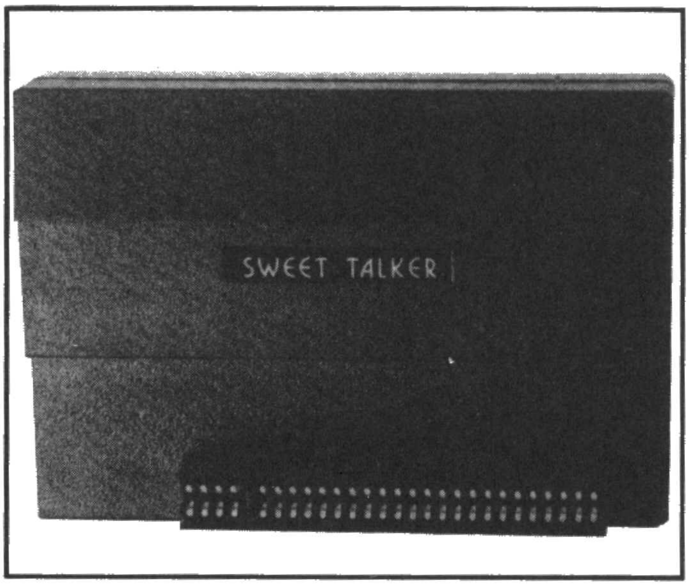
1 Select EPROM type.
2 Transfer disk to buffer.
3 Transfer buffer to disk.
4 Transfer buffer to EPROM.
5 Transfer EPROM to buffer.
6 Verify EPROM against buffer.
7 Formatted dump of buffer.
8 Load area of buffer with character.
9 Change one buffer location.
10 Exit.

It takes a long time to program these large EPROMs during option 4 and it was found desirable to give some indication of progress on the screen, to reassure the user that the system was in fact working! To this end the screen displays a dot for each page (256 bytes) to be programmed and as each page is completed the dot is overwritten with a character A. Once the EPROM is programmed it is wise to use option 6 to verify that the EPROM has been programmed correctly.

Continued next month.

SWEET TALKER

Ken Alexander lends an ear to Cheetah's Spectrum Speech Synthesiser.



Cheetah's Sweet Talker is an elegant implementation of the allophone approach to speech synthesis. The unit consists of a black plastic box that plugs into the expansion connector on the rear edge of the Spectrum (in common with other add-ons the Sweet Talker makes all these connections available on a PCB 'plug' on the rear of the unit in order that other add-ons can be used in addition to the Sweet Talker). The rear of the unit also features a speaker through which the synthesised speech is reproduced.

The unit is supplied with a brief, yet informative, three page instruction manual and a cassette that introduces the facilities offered by the Sweet Talker in the best possible fashion – by demonstration.

Before going on to describe the module in any more detail, an explanation of the allophone method of speech synthesis is probably in order.

Space Saving Technique

There are two major approaches to the electronic synthesis/generation of speech. The first, as adopted by Acorn for the BBC computer, is to store a digitised recording of the words/phrases to be uttered in ROM memory. In fact producing these ROMs is not quite as straightforward as storing the output from an A/D converter fed with the audio signal to be processed. The signal is subjected to various processes designed to reduce the amount of memory required to store the speech. The major drawback with this technique of speech generation is that the words capable of being produced are

Table 1. The allophone's that the Sweet Talker can produce together with examples of their use are adequately described in the unit's brief but informative manual.

SILENCE DURATION
10ms
30ms
50ms
100ms
200ms

LETTER	ALLOPHONE	EXAMPLES
A	*AE	extract, acting
	*AO	talking song
	*AX	lapel
	EY	great, statement, tray
	AR	farm, alarm, garment
	XR	hair, declare, stare
B	BB1	final position: rib; between vowels; fibber; in clusters; bleed, brown
	BB2	intial position before a vowel: beast
C	CH	church, feature, cheetah
D	DD1	final position: played, end
	DD2	initial position: down; clusters: drain
E	*EH	extent, gentlemen
	IY	treat, people, penny
	ER1	letter, furniture, interrupt
	ER2	Monosyllables; bird, fern, burn
	YR	hear, earring, irresponsible
F	*FF	may be doubled for initial position and used singly in final position; food
G	GG1	before high front vowels: YR, IY, IH, EY, EH, XR, guest
	GG2	before high back vowels: UW, UH, OW, OY, AX: and clusters; green, glue
	GG3	before low vowels: AE, AW, AY, AR, AA, AO, OR, ER; in medial clusters; anger; and final position; peg
H	HH1	before front vowels: YR, IY, LH, EY, EH, XR, AE, HE
	HH2	before back vowels: UW, UH, OW, OY, AQ, OR, AR: HOE
I	*IH	sitting, stranded
	AY	kite, sky, mighty
J	JH	judge, injure
K	KK1	before front vowels: YR, IY, IH, EY, EH, XR, AY, AE, ER, AX; initial clusters: cute, clown, scream
	KK2	final position: speak; final clusters: task
	KK3	before back vowels: UW, UH, OW, OY, OR, AR, AO: initial clusters: crane, quick, clown, scream

LETTER	ALLOPHONE	DATA	EXAMPLES
L	LL	45	like, hello, steel
	EL	62	little, angle, gentlemen
M	MM	16	milk, alarm, ample
N	NN1	11	before front and central vowels: YR, IY, IH, EY, EH, XR, AE, ER, AX, AW, AY, UW; final clusters; earn
	NN2	56	before back vowels: UH, OW, OY, OR, AR, AA: No
	NG	44	string, anger
O	*UH	30	cookie
	*AA	24	pottery, cotton
	OY	5	noise, toy, voice
	UW2	31	in monosyllabic words: two, feed
	OW	53	zone, close, snow
	AW	32	sound, mouse, down
	OR	58	fortune, adorn, store
	*AO	23	talking song
P	PP	9	pleasure, ample, trip
Q	KK3	8	before back vowels: UW, UH, OW, OY, OR, AR, AO: initial clusters; crane, quick, clown, scream
R	RR1	14	initial position: read, write, x-ray
	RR2	39	initial clusters: brown, crane, grease
S	SS	55	may be doubled for initial position and used singly in final position: vest
	SH	37	shirt, leash, nation
T	DH1	18	word-initial position: this, then, they
	DH2	54	word-final and between vowels: bathe, bathing
	*TH	29	may be doubled for initial position and used singly in final position: thin
	TT1	17	final clusters before SS: tests, its
	TT2	13	all other positions: test, street
U	UW1	22	after clusters with YY: computer
	*UH	30	full
	*AX	15	instruct
	ER2	52	monosyllables; burn
	VV	35	vest, prove, even
W	WW	46	we, warrant, linguist
Y	YY1	49	clusters; cute, beauty, computer
	YY2	25	initial position; yes, yarn, yo-yo
Z	ZZ	43	zoo, phase
	ZH	38	beige, pleasure

*These allophones can be doubled.

determined by the manufacturer of the speech ROM. To counter this disadvantage, the quality obtainable from a ROM based speech system is very high and some are capable of producing output that is very close to the sound of the human voice.

In order to overcome the restricted vocabularly of a ROM system it is possible, with a fair degree of skill, to 'sound' only part of two or more words and thus build up a new word from sections of those that are available. This sort of technique can extend the scope of such systems and gives a clue to the allophone technique.

The Allophone Technique

The term refers to the individual sounds that go to make up a particular word. **Table 1** shows the set of sounds that the Sweet Talker is capable of producing together with a list of words which feature the various allophones.

To build up a particular word it is necessary to select a series of these allophones and sound them in quick succession. From this it is apparent that an allophone system demands far more programmer overhead than a ROM system and that the quality of the final result depends very much on the skill with which the programmer can break down the word to be produced into a set of allophones.

Speakers' Corner

The Sweet Talker is straightforward to use and, as it draws its power from the Spectrum's PSU and uses its internal speaker to reproduce the speech, setting it up is merely a matter of plugging it into the Spectrum's expansion port.

Loading the software supplied on the demonstration cassette will produce a welcome message on the TV screen as well as 'spoken' instructions. The audio output gives a good indication of the quality of speech obtainable with skillful programming of the unit.

The speech is understandable but not as good as the quality obtainable from ROM based systems – the voice is flat and lacks any intonation. It is very definitely computer speech and could not be confused with the sound of a human voice.

To sound a particular allophone the Spectrum's OUT command is used in conjuction with a DATA statement. For example, to produce the word 'Cheetah' the following program could be used.

```
Example 'Cheetah'
10 DATA 50,19,13,15,15,0
20 FOR I = 1 TO 6
30 READ A
40 OUT 7, A
50 NEXT I
RUN
```

The listing for the demonstration tape can be examined and will give the word patterns for many more words. It would however have been of great help if the instruction book had given data strings for some common words, the numerics one to ten and such words as on/off, up/down, left/right for example.

Last Word

The Sweet Talker is a well made product that is quite straightforward to use. The quality of speech produced is not up to the standard of ROM based systems such as that used by Acorn but the Sweet Talker does not suffer from any restriction of vocabularly that are evident in all ROM speech synthesisers.

With care at the programming stage any word that is required can be built up from the available allophones and the result will be understandable if not human sounding. The Sweet Talker is an ideal low cost introduction to speech generation techniques.

Cheetah Marketing
359 The Strand
London
WC2R 0HS

E&CM

SINGLE CHIP MICROCONTROLLER

Richard Whitlock describes the design of a one chip microcomputer that can be built at a realistic price. Low cost programming hardware and a BASIC cross-assembler all go to make this design the first such controller truly suitable for one off applications.

There are many one-chip microcomputers on the market, all aimed at the "low-end" user, the manufacturer of "smart" toys and games, "intelligent" cookers and washing machines, automobile and industrial control and monitoring systems, and measuring instruments from micrometers to the butcher's scales. Why does Motorola's recent entry to the 8-bit class of this market, the MC6805, warrant more than cursory examination in a magazine such as this, when other devices go largely unnoticed? The answer lies in the fact that, with the introduction of this family of devices, Motorola have opened the door to true one-off, one-chip logic designs at a realistic price.

Masking The Problems

Previous one-chip microcomputer systems have clearly been aimed at the large volume user, with little or no thought being given to the need of the small volume user. Here the requirements are for a simple to program, cheap to buy, highly versatile circuit element which does not require large expenditure, or heavy commitment to future expenditure, to obtain the modest numbers required. Mask programmed microcomputers can be bought for about the same price as their conventional bus orientated counterparts if sufficient numbers of a single desing are ordered, but mask processing charges are horrific on a small production run. Admittedly, there have been a number of prototyping systems produced which could physically be fitted into the same socket as a custom programmed, mask-ROM device, but they have generally been too expensive or awkward to program to represent a true one-off or small volume solution.

In some cases all that has been offered has been an in-circuit emulator, a device which interfaces both to the target system and to a host computer from which it receives its program and to which it sends a constant stream of data about its internal states for use in program modification at the design stage. Such systems are clearly useless for anything but developing an application program. A second approach has been to supply a special version of the one-chip microcomputer with no on-board ROM to store the user's program, having instead either additional pins or an integral IC socket, upon which an address and data bus are generated, allowing the connection of a conventional EPROM. Costs have tended to be high due to low volumes sold, since the devices represented little advantage over the three chip minimum systems supported by most microprocessor families without any need to resort to mask programming, (i.e. microprocessor, RAM/I/O/Timer IC and EPROM), and thus did not attract the small volume user to swell sales. Where a true EPROM based version of the one-chip microcomputer is available it is difficult to program, typically requiring a manufacturer's development system plus a special interface and program (typically several K£!). This again is not an inviting prospect for the small first-time user of a device, and absolutely prohibitive for one-off users such as hobbyists.

Features	MC68705P3	MC68705R3	MC68705U3
Technology	HMOS	HMOS	HMOS
Number of Pins	28	40	40
On-Chip RAM (Bytes)	112	112	112
On-Chip User ROM (Bytes)	1.8K EPROM	3.8K EPROM	3.8K EPROM
External Bus	None	None	None
Bidirectional I/O Lines	20	24	24
Unidirectional I/O Lines	None	6 Inputs	8 Inputs
Other I/O Features	Timer	Timer, A/D	Timer
External Interrupt Inputs	1	2	2
EPROM Version	–	–	–
STOP and WAIT	No	No	No

Table 1. The 6805 family of devices features three true EPROM versions providing a variety of facilities in either a 28 or 40 package.

Problems Solved

As can be seen from **Table 1,** the MC6805 family falls into the third category mentioned above, at first glance, having three true EPROM based versions which cover most of the facilities offered by the remaining mask programmed HMOS versions. However, when it comes to programming, the new Motorola chips are in a class of their own. They program themselves! (No, they don't write their own source code and assemble it, but they do take care of much of the rest of the process). When placed in a very straightforward circuit they program their own EPROM, from a user programmed EPROM, completely automatically. Thus, if the would-be one-chip microcomputer user is already equipped to program a 2532, they can provide themselves with the additional hardware for 68705 programming for well under £5. Well within the reach of the most impecunious hobbyist.

This month and in subsdequent articles we will aim to provide all the information, circuit details and board patterns necessary to start the reader on the way to custom one-chip logic systems, be they stand-alone applications or add-ons for existing computers. A cross-assembler program, written in 8K Microsoft BASIC for easy transportability between machines, is planned and assembly language routines are already written for implementing many standard control and communication functions as well as data gathering and processing operations.

Structure And Functioning

Figure 1 shows the borad outlines of the internal structure of the 68705. **Fig 2** shows the pin assignments for the device.
each of the three variants.

We will first consider the features and functions that are common to all three variants, which are, roughly speaking, those embodied in the 28-pin MC68705P3, and ▶

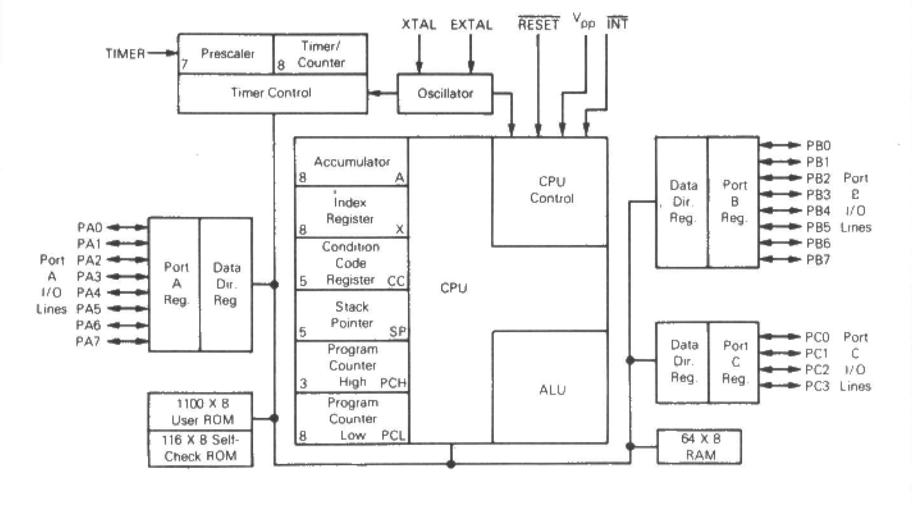


Figure 1. The broad outline of the basic 68705P3 member of the family.

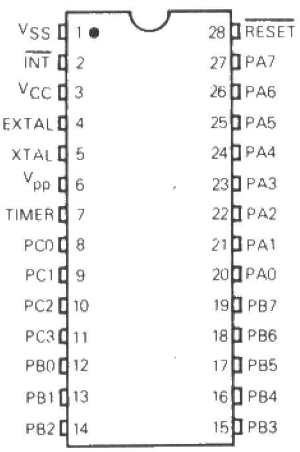


Figure 2. Pin-out of the 68705P3 MCU.

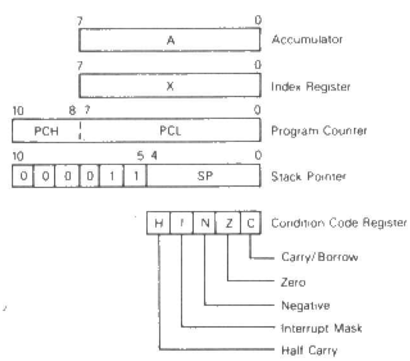


Figure 3. Details of the processor registers. These should look familiar to anyone with experience of either 6800 or 6502 machine code programming.

then move on to the additional capabilities offered by the 40-pin MC68705U3 and MC68705R3.

The Processor

Unlike the Intel 8048 or the Zilog Z8, the Motorola 68705's processor is implemented completely separately from the memory and I/O registers, so there is no need for the programmer to keep in mind which bank of RAM registers is currently close-coupled to the Arithmetic and Logic Unit.

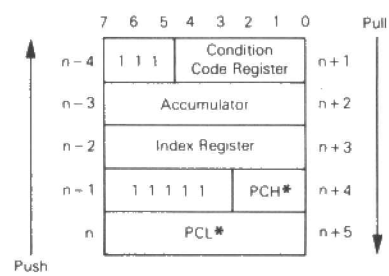
The processor registers (see **Fig 3**) are quite conventional in their operation and users with experience of machine code programming on either the 6800 or the 6502 should find little to puzzle them. Details are as follows:-

Accumulator (A) – A general purpose 8-bit register used to hold operands and results of arithmetic and logic operations and other data manipulations.

Index Register (X) – An 8-bit register whose primary function is to hold a value which, when added to a value specified in a program instruction creates an effective address dependant upon past conditions in the system, thus allowing a single programmed routine to cover a number of particular cases. In the 68705s the Index Register can also be used to perform data manipulations using the Read/Modify/Write group of instructions, (see INSTRUCTION SET below).

Program Counter (PC) – A register than normally contains the address of the next instruction byte to be fetched and executed. In the 68705P3 which has a total memory space of 2K addresses (2048 locations) the Program Counter is an 11-bit register. In the 68705U3 and 68705R3, which have 4K of memory space (4096 locations) the Program Counter is a 12-bit register.

Stack Pointer (SP) – A register that contains the address of the next free location on the processor stack. At Processor Reset, or on execution of a Reset Stack Pointer instruction, the Stack Pointer is set to point to the highest RAM address, 07FH. From there it is decremented as data is pushed onto the stack and incremented again as data is pulled off the stack. Only the lower 5 bits of the Stack Pointer are variable, giving a maximum stack depth of 31 locations, i.e. down to address 061H. The upper bits of the Stack Pointer are permanently set. In the 68705P3 the Stack Pointer is an 11-bit register and in the 68705U3 and 68705R3 it is a 12-bit register. The stack area is used during the processing of interrupts and subroutine calls to save the previous contents of all or some of the processor registers. The register contents are pushed onto the stack in the order shown in **Fig 4.** An interrupt results in all the processor registers, except the Stack Pointer itself, being pushed onto the stack, while a subroutine call results in only the Program Counter being stacked. The maximum depth of 31 locations for the stack restricts subroutine nesting depth to 15 levels if interrupts are not allowed, and less if they are allowed.



*For subroutine calls, only PCL and PCH are stacked.

Figure 4. The register contents are pushed onto the stack in the order shown when an interrupt is detected.

Condition Code Register (CC) – A 5-bit register in which four of the bits are used to indicate significant results of the last instruction executed. The fifth bit is used to control the reception of interrupt requests from other parts of the system. The four result flags can each be tested by program instructions and specific action taken as a result of their state. Each of the five bits is explained below.

Half Carry (H) – Set (to logic '1') during addition instructions to indicate that a carry has occurred between bits 3 and 4 of the result. This indication is necessary for arithmetic operations where two BCD digits

are encoded within a single byte.

Interrupt (I) – Set at Reset and subsequently cleared (to logic '0') and set under program control, this bit, when set, prevents all interrupt requests from affecting the processor with the exception of the software interrupt instruction, which is executed like any other program instruction. An interrupt request reaching the processor while the Interrupt bit is set is latched until such time as the bit is cleared, when it becomes operative.

Negative (N) – When set, this bit indicates that the result of the last arithmetic, logical or data manipulation was negative, i.e. bit 7 of the result was logic '1'.

Zero (Z) – When set, this bit indicates that the result of the last arithmetic, logical or data manipulation was zero, i.e. all bits of the result were logic '0'.

Carry/Borrow (C) – When set, this bit shows that a crry or borrow occurred at bit 7 of the result of the last operation executed. This bit is affected during arithmetic, bit test and branch, shift and rotate operations.

The Memory

The memory map of the 2K 68705 variant is shown in **Fig 5.** The first 16 addresses are more or less fully occupied by the I/O port, control and status registers, depending upon the variant in question. Above these every 68705 has a block of 112 continuous RAM bytes, shared by the user program and the processor's stack, as mentioned above. From address 080H upwards is the main user EPROM area where the particular application program must be stored. In the 68705P3 this area is 1796 bytes in extent, while in the 68705U3 and 68705R3 it is 3768 bytes in extent. Note that 128 bytes of the user EPROM are within page zero (the area of memory whose addresses can be held in an 8-bit register) and thus, together with all of the RAM and I/O, can be referenced using short form "direct addressing" ("zero-page addressing" in 6502 parlance). Subroutines called from many points in the main program, frequently used data tables etc. can profitably be located in these 128 bytes to take advantage of the shorter and faster instructions capable of reaching them. Above the main user EPROM area is a single location known as the Mask Option Register (MOR). The MOR is again an EPROM register whose contents must be specified by the user. The function of the various bits in the MOR is to carry information used at Reset to configure various parts of the on-chip circuitry.

Greater detail of its operation will be given next month when we discuss the MASK OPTION REGISTER. Above the MOR again is the "bootstrap" ROM containing the self-programming routines which, when activated, enable the 68705 to program its own EPROM from an external user programmed EPROM.

The last 8 bytes of the memory space are again user defined EPROM locations, which must be programmed with the Interrupt and Reset vectors required for the particular application.

Next month we'll continue the description of the facilities offered by the 68705 but to round off this month we'll whet the appetite by showing just how straightforward it is to program the MPU.

Self-Programming Process

The 68705s have a block of mask programmed ROM containing a "bootstrap" program which is used to program the on-board EPROM with the user's program. At the top of this mask programmed ROM block, immediately below the user defined interrupt service and Reset vectors, is a further vector which is fetched only when the conditions required for programming are all met. The mask programmed contents of these two locations direct the processor to the start of the self-programming routine.

During the execution of this routine the master oscillator is forced into the crystal mode of operation, regardless of the content to be programmed into the CLK bit of the MOR. For the timing of the programming operations to be correct a 1 MHz crystal must be used to regulate the master oscillator.

The self-programming routines control the actual programming of the EPROM bit cells by manipulating the least significant two bits of the Programming Control Register at location 00BH. These two bits enable the latching of the data to be programmed onto the outputs of the EPROM block and the generation of the high voltage programming pulses which drive electrons "through" the insulation surrounding the isolated gates of selected transistors within the EPROM bit cells, leaving a semi-permanent charge on the gate and forcing the transistor into a conducting state, thereby pulling the output of the cell up to logic "1" from its unprogrammed logic "0" state.

During normal operation the Programming Control Register is addressable and bit 0 may even be set and cleared by the processor, but it has no effect upon the operation of the EPROM so long as bit 2 is set, and bit 2 is read only, being cleared only by the presence of a high voltage on the V_{PP} pin, so there is no danger of accidentally entering the self-programming mode from a faulty user program. V_{PP} is connected to V_{CC}, the nominal 5 V supply, during normal operation. **Fig 6** shows the Motorola recommended circuits for programming both the 2K and the 4K memory space versions of the 68705.

Note that the pattern EPROM from which the 68705 programs itself must contain an exact duplicate of the information that is to be programmed into the user EPROM of the 68705. However locations which corres- ▶

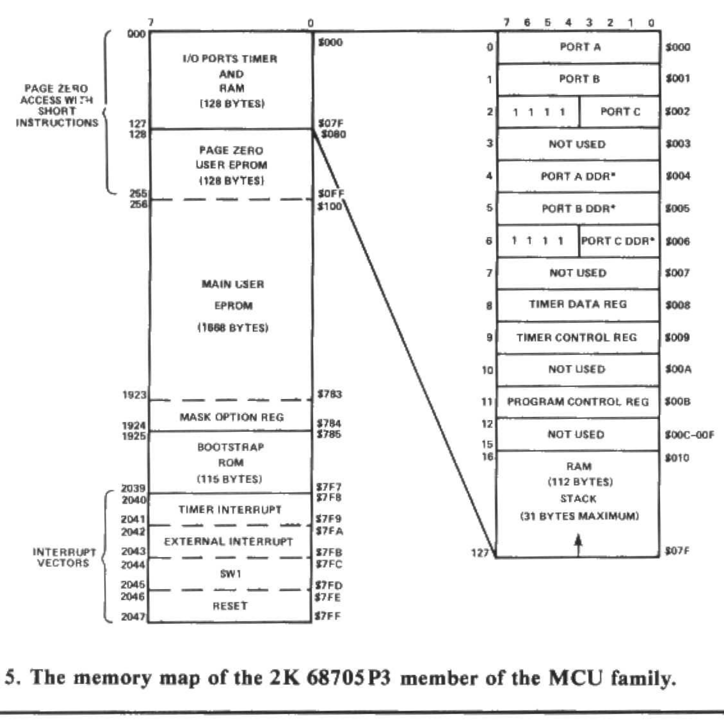


Figure 5. The memory map of the 2K 68705P3 member of the MCU family.

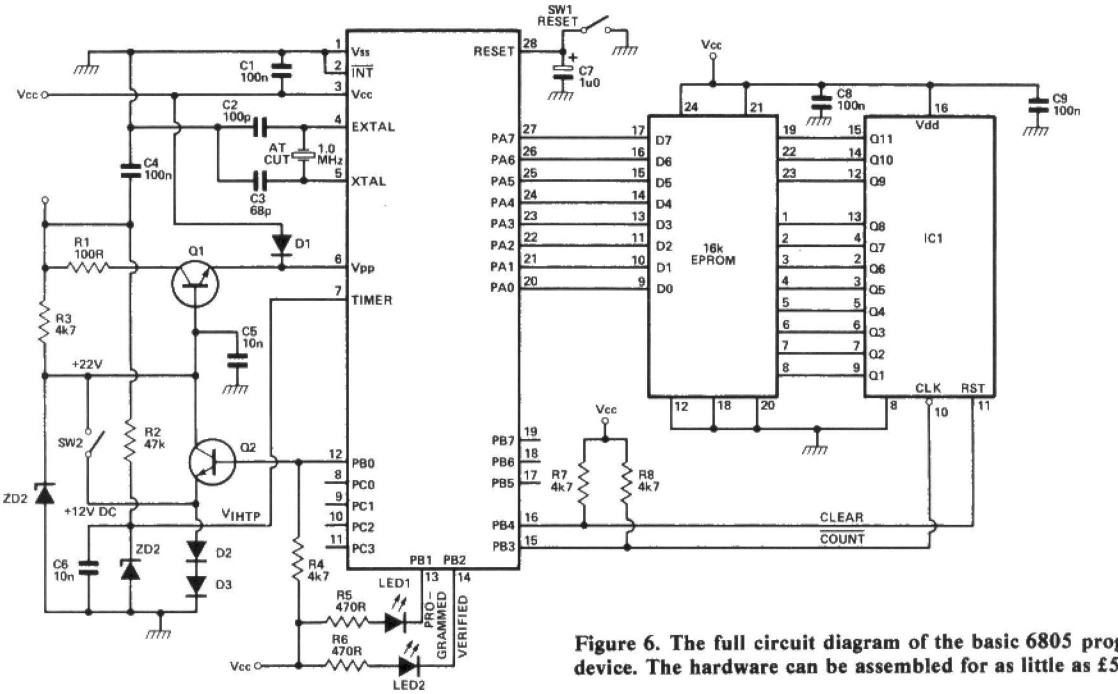
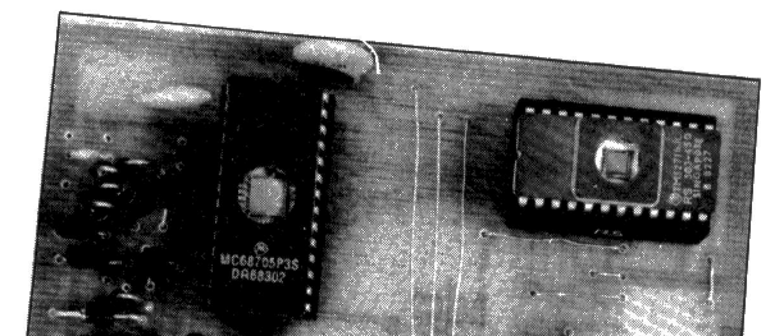


Figure 6. The full circuit diagram of the basic 6805 programming device. The hardware can be assembled for as little as £5.00.

pond to non-EPROM locations in the 68705 are ignored during the self-programming process and can be left unprogrammed. A corollary of this fact is that it is not necessary to use a 2716 to carry the data to be programmed into a 2K memory space 68705, a 2532 with the data located in the lower half will do just as well. A 2532 is used to carry the data for both 2K and 4K versions of the 68705.



The operation of the programming board is quite simple. Before inserting the 68705 and the 2532 into their respective sockets, ensure that switches S1 and S2 are closed and switches S3 and S4 are open. When the ICs are safely socketed, close S4 to apply the 5V supply, close S3 to apply the 26V programming voltage, open S2 to apply the high voltage Program Mode Vector Fetch signal to the TIMER pin, and finally open S1 to remove Reset.

On emerging from the Reset condition the 68705 configures Port B as outputs and then pulses PB4 to reset the 4040 CMOS counter and then clocks it up to a count of 080H (128) by pulsing PB3. The counter outputs are now addressing the data for the lowest user EPROM location in the 68705, stored in the 2532. This data is read in via Port A and latched onto the user EPROM outputs from where it is "burned" into the EPROM cells just like the operation in a normal EPROM programmer. The counter is advanced and the process repeated until all locations up to and including the MOR have been programmed. Then a further burst of pulses on PB3 step the counter on past the "bootstrap" area to program the last eight bytes, the Interrupt service and Reset vectors. The "Programmed" LED is the lit.

The whole process is then apparently repeated, but in fact a "verify" operation is undertaken, comparing data read from Port A with the newly programmed contents of the user EPROM. If all is well the "Verified" LED will light up and the operation is complete. To power-down the board, before removing the 68705, first clost switch S2, then S1 and finally open S3 and then S4.

Note that the order of opening and closing switched at the beginning and the end of the programming process is to be strictly adhered to, or the 68705 may be destroyed!

E&CM

Next Month – further details of the 68705's facilities plus construction details of the programming hardware.

Part 1 NEW SERIES

PRACTICAL ELECTRONIC TECHNIQUES

A new series that will introduce various aspects of the design and construction of electronic circuitry. This month Brian Lovell sheds light on what to many people is a black art – the design and production of Printed Circuit Boards.

The prototype unit of many designs will probably consist of a fairly hacked about piece of verboard with a birds nest of untidy wiring between the various components. To transform a circuit in this state into a neat and reliable unit the most effective method is to prepare a printed circuit board layout from the circuit schematic.

What follows is a description of the design and manufacture of PC boards together with a list of suppliers for the materials required in the production of the boards. The initial outlay in equipment is not too great and, with a little practice, anyone following the advice given below should be able to turn out single sided boards quickly and cheaply.

Let Us Begin

Bearing in mind that these instructions are for single side boards the following design limits should be noted.

1 Have no more than three tracks running underneath an IC.

2 Have no tracks running between the pins of an IC except for the ends where 1 holds.

3 Don't take tracks closer than ¼" to the edge of the board.

4 The inter-track capacitance must be taken into consideration for high frequency tuning circuits.

5 Wire jumpers can be used on the component side of the board if necessary.

6 Design the circuit from the component side. All chip pin designations are given in this format. Having two designs, one the mirror image of the other, is bound to cause confusion later on.

Spend plenty of time on this section as mistakes are awkward and time consuming to rectify later on. Start by designing small boards or break a large circuit into small sections (obvious partions include separating the power supply and keeping inputs separate from outputs). This is particularly important when tracking out DC amplifiers. In digital circuits CMOS is relatively immune from noise but don't forget to use 100n tantalum capacitors on V+ near the chips.

1st Stage

Most circuits use ICs these days, the 'legs' being on a 0.1" matrix. It is convenient to design all circuits on such a matrix. Buy an accurate grid (Ambit), 10 lines to the inch. Tape light paper or tracing paper over this grid and reduce your circuit down to the correct size. Use a pencil and have a good rubber ready, the plastic ones are best. Once again take your time and check your work. Space the strips evenly and follow normal design rules.

Sharp corners encourage peeling and hair lines of copper and solder. Large areas of copper have no advantage in low current circuits and can act as a heat sink during the soldering process leading to poor solder connections.

2nd Stage

Preparation of the master. This involves transferring the pencilled pattern onto polyester film using stick on symbols and track. These are high density and jet black for the photographic process later on. Black ink will simply not work. The most well known of these kits is the instant lettering system by Letraset, however they do not reach the requirements for electronics. The best system discovered to date is produced by Alfac. The symbols required are sold in small blister packs of 5 sheets each. These retail at about £2.20 per pack or 40 to 50p per sheet. It is not easy to find but some electronic shops stock it including Ambit. It is best to use a 1mm track width, any smaller and undercutting can occur during the etching process. A list of the most useful packs is given in **Table 1.**

E.C. 942 straight track

E.C.953/1 curves, quarter circles

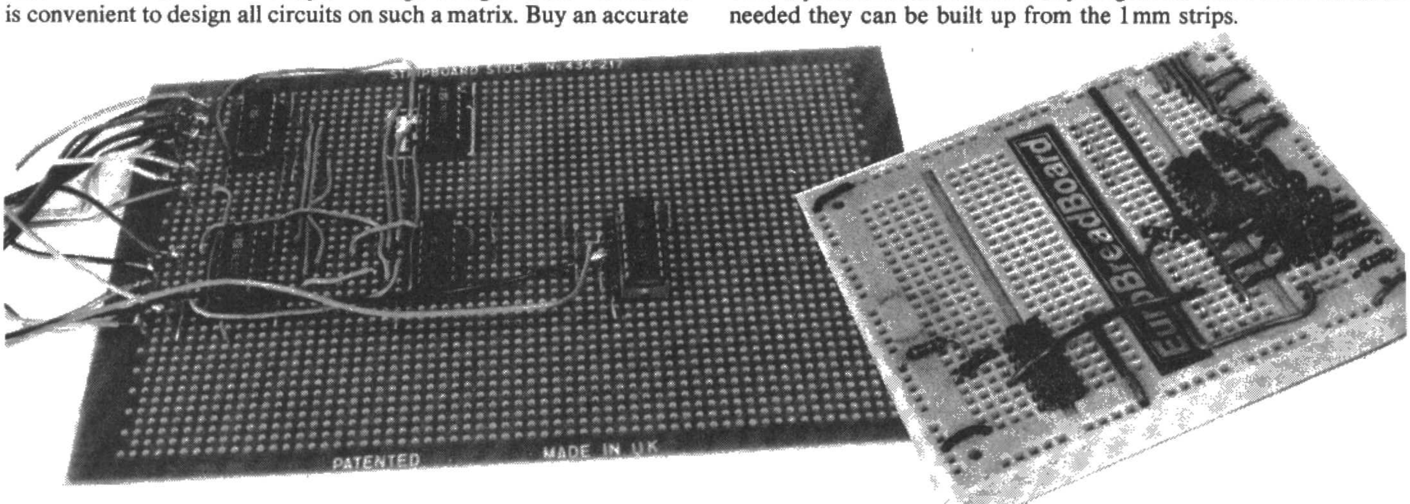
E.C.927/1 T intersections

E.C. 956/1 short bends

E.C. 994/1 chip pads

E.C. 996/1 chip pads

Tracks can also be bought in rolls. They are much thicker than ordinary transfers and unnecessary for general use. If wider tracks are needed they can be built up from the 1mm strips.



The completed prototype of many designs usually looks something like these projects. Whether some form of solderless system or veroboard is used the finished result is unlikely to look tidy nor to be capable of reliable operation over a period of time.

The base that these transfers are pressed onto must be chosen with care. It must be dimensionally stable, it does not have to be completely transparent since only contact prints are made. Acetate is not suitable since it curls with use and changes shape. Polyester film is ideal (Ambit). An excellent alternative is Selectatrace from Selectasine, one side is rough improving adhesion of the transfers, it can be bought in A1 sized sheets for about £1.20.

Position the polyester above the gride, rough side up and fasten with masking tape to prevent movement. Transfer the pencil pattern to the polyester remembering to preserve correct spacing. A surgeons knife or scalpel is ideal for prising wrongly placed transfers from the polyester. It may help to lightly pencil in the pattern on the polyester first, but as this reduces adhesion it must be done sparingly. A blunt knitting needle is a good tool for applying the transfers, gently at first to prevent cracking and then harder to ensure a firm placing. It helps to do all of this on a clean sheet of cardboard. Despite the small size of the Alfac blister sheets and transfers not in use pick up dirt rendering them useless if care is not taken. Grease on the polyester doesn't help either so keep those hands clean and dry, the fastidious could even use cotton gloves. After completion and checking keep the master in an envelope away from dust.

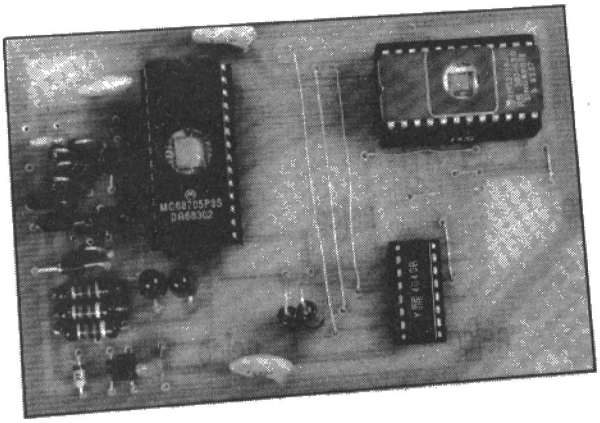
3rd Stage

The master can now be used with pre-coated copper board to prepare the finished design. The technique requires a method for transferring the master pattern to the board. This is done photographically, using Ultra Violet (UV) light. Commercial kits for this task are available but are expensive starting at £35. A cheaper and entirely satisfactory method is to make your own. Use a 300W UV Edison screw bulb, £10 and brass socket which is necessary because of the heat generated. This can be hung above the work using the power cord. Depending on the size of your work, 24" is satisfactory giving an exposure time of about 12 minutes. A sheet of thin glass is also needed to keep the polyester flat. Its UV absorptive properties are allowed for in the exposure time.

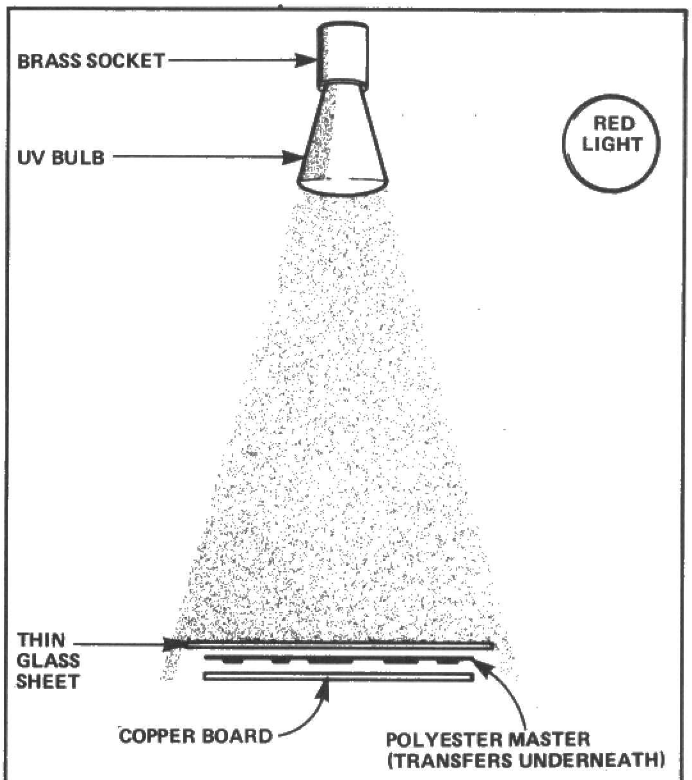
Pre-coated Board

This can be bought from a variety of sources Ambit Maplin etc. Handle as for photographic paper. Use a red darkroom lamp throughout the exposing and developing process. Under the red light carefully peel off the plastic and place centrally, copper side up under the UV lamp. Now place the master, transfer side down on the copper board. Cover with the well cleaned and dust free glass sheet. Expose for approx 12 minutes. **Warning,** *UV light can damage the eyes, wear sunglasses and do not look directly at the bulb.* It is better to overexpose than underexpose, trials may be necessary.

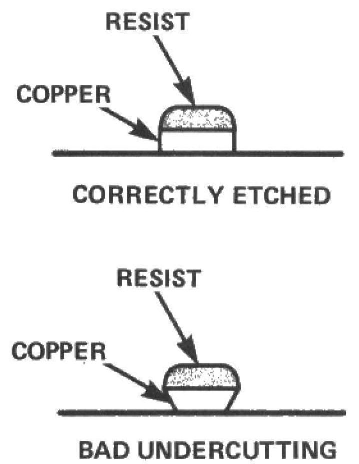
When exposed the board is developed in a sodium hydroxide solution at 20°C. A tablespoon of crystals in 500ml of water is about right. A photographic developing tray is handy here, wear plastic gloves because sodium hydroxide is nasty stuff. When, after a few minutes the image appears, wash off in running water delicately removing the unwanted material with your finger. The pattern left is



By designing a PCB for the final circuit a neat and reliable unit can be produced.



The third stage of the PCB production process involves exposing a coated board to UV light in order to transfer the master artwork.



Care should be taken when etching the PCB. If etching is not completed quickly it is possible to badly uncut the tracks.

an etch resist positive. Unwanted areas of copper are unprotected by the resist. Leave to dry.

Etching

A ferric chloride solution is used for this. Dissolve the crystals in warm water. The crystals can be bought from Maplin, various electronic suppliers or in bulk from chemical firms.

To prevent undercutting etching must be done quickly. Flat trays can be used but it is best to force the process. Add a little HCL (Hydrochloric Acid) to the ferric chloride solution. Pour in a large pyres beaker large enough to hold the copper board vertically and heat it up to 40°C. Pump air into the beaker with a fish tank air pump. It is possible to etch in about 2 minutes using this method. Wash in running water and remove the etch resist with screen cleaner or any alkaline solvent. ▶

Drilling the Board

Use a high speed drill. A pin vice may be necessary to hold the small bit (£3). 1mm is ideal for chip legs and general components.

Copper Board (Laminate)

Either paper based with a phenolic resin or fibre glass. The former are unsuitable for plated through holes and restricted to single sided PCBs. Fibre glass boards are about 50% more expensive and much stronger. All double sided and multi-layered PCBs are made of this material. Fibre glass boards cannot be punched and wear out drill bits readily. 1/16" (or 1.6mm) is the most common thickness for both laminates. 1oz board is quite sufficient for low current use and doubles the life of your etch bath. Both laminates can be sheared using a guillotine, small rake angles (the angle between the two blades) prevent the laminate from splitting (5° to 10°). Alternatively a circular saw or band saw can be used. Fibre glass boards destroy normal cutting edges on both. Bonded diamond cutting edges solves the problem at a price. This plus the problem of fibre glass dust makes shearing the best method. PCB manufacturers will sell in quantity at approx half the price of general electronic suppliers (Oval Products).

List Of Suppliers

All of the prices should be checked before ordering.

Oval Products,
Benedict road,
Mitcham,
Surrey,
London CR4 3UQ.
(01) 648 5913

Pelltech Ltd (Alfac agents)
Station Lane,
Witney,
Oxon OX8 6YS
(0993) 72130
They will give you the address of the nearest Alfac Stockist

Selectasine,
22 Bulstrode Street,
London W1M 5FR
(01) 935 0768

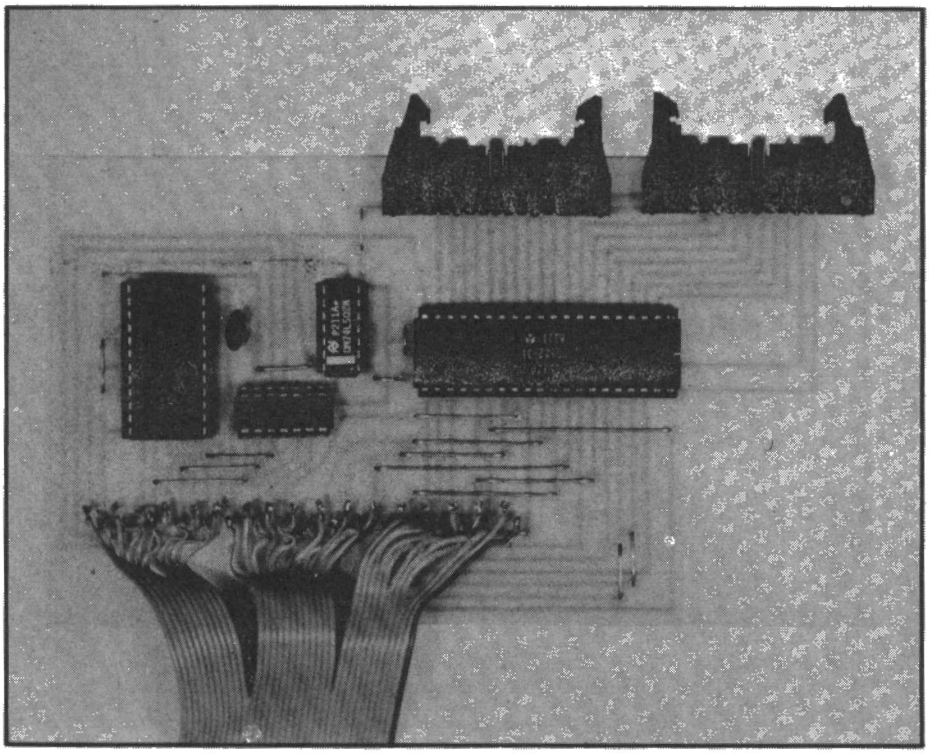
Sericol Group Ltd.,
24 Parsons Green Lane,
London SW6 4HT
(01) 736 3388

Taylor V. M. Electrical Ltd.,
High Holburn
(near Chancery Lane tube)
(01) 405 0042
Stock Edison screw brass sockets.

Wholesale Fittings,
13 Berners Street,
London W1P 4BY
(01) 637 5393
Stock UV lamps, Edison screw 300W.

Ambit International,
200 North Service road,
Brentwood,
Essex CM14 4SG
(0277) 230 909

R.S. Components
(Wholesale and account holders only)
Enquiries (01) 250 4000



ORIC 6522 IN/OUT PORT

Robert Penfolds describes a sophisticated input/output port for the Oric computer.

In common with other low cost microcomputers the Oric 1 has only a limited range of built-in interfaces, and does not feature a port which can be used directly as a latched output to drive LEDs, relays etc., or as an input for switches or analogue to digital converters. It is possible to add a latch to the parallel printer port for simple output applications, as described in an earlier article in *E&CM*, but for more demanding applications a PIA or VIA can be added to the expansion port.

The latter has the full complement of address, data, and control bus lines, and it would presumably be possible to add any of the normal 6800/6502 family of interface devices here. The 6522 was chosen as it is relatively straightforward to use even if only one or two input or output ports are all that is required. In addition it has a number of features which ultimately make the device more versatile than alternatives such as the 6821.

The device provides two 8 bit input/output ports which have each bit individually programmable as an input or an output. Additionally, each port has two handshake lines, one of which can be used as in input or an output, and one which is an input only. The device also has two 16 bit counter/timers, and it provides the opportunity to use interrupts. In both cases these facilities have not been tried in earnest by the author, but there is no obvious reason why they should not be usable in specialised applications which require their use. They can only be employed in conjunction with machine code routines however and their use requires an in-depth understanding of both the Oric and the 6522. In this article we will only consider the use of the interface as an input/output port.

Using The Ports

All the registers of the 6522 together with their addresses are shown in **Table 1,** but only the first four are used when using the interface as simple input/output ports.

An interesting feature is the inclusion of two registers for Port A, one which gives operation of the handshake lines (if they are used), and one which does not. In practice Port A would normally be addressed at BFF1 whether handshaking was required or not. The point about register BFFF is that it gives the option of using Port A without activating the handshake lines and at the same time using this port with handshaking via BFF0. This is not something that is likely to be used a great deal in practice, and we will assume here that Port A is always accessed via BFF1.

Before a Port can be used it is necessary to set up each data line as an input or an output, as required. This is where the Data Direction Registers (DDRs) at BFF2 and BFF3 are used. Setting a bit of these registers at 1 sets its corresponding data line as an output, while setting a bit at 0 makes the relevant data line an input. For instance, at switch-on the DDRs are both set to zero and all the data lines are inputs, but if 240 (11110000 in binary) is sent to DDRA at BFF3, PA4 to PA7 are set as outputs while PA0 to PA3 are left as inputs. In most cases all lines of a port will be set as inputs or outputs, and it is then just a matter of writing 255 to the appropriate DDR if an output port is required, or, alternatively, writing 0 to the DDR to ensure that all lines have been properly set as inputs.

In general there is not likely to be any difficulty in driving opto-isolators, logic ICs, relay drivers, and so on from either port, or in driving the ports from logic ICs, A/D

. . . Turn to Page 50

HEX	DECIMAL	REGISTER
BFF0	49136	Port B
BFF1	49137	Port A (handshake)
BFF2	49138	Data Direction Register B
BFF3	49139	Data Direction Register A
BFF4	49140	Timer 1 Counter (low byte)
BFF5	49141	Timer 1 Counter (high byte)
BFF5	49142	Timer 1 Latch (low byte)
BFF6	49143	Timer 1 Latch (low byte)
BFF8	49144	Timer 2 Counter (low byte)
BFF9	49145	Timer 2 Counter (high byte)
BFFA	49146	Serial Shift Register
BFFB	49147	Auxiliary Control Register
BFFC	49148	Peripheral Control Register
BFFD	49149	Interrupt Flag Register
BFFE	49150	Interrupt Enable Register
BFFF	49150	Port A (no handshake)

Table 1. 6522 registers together with associated addresses.

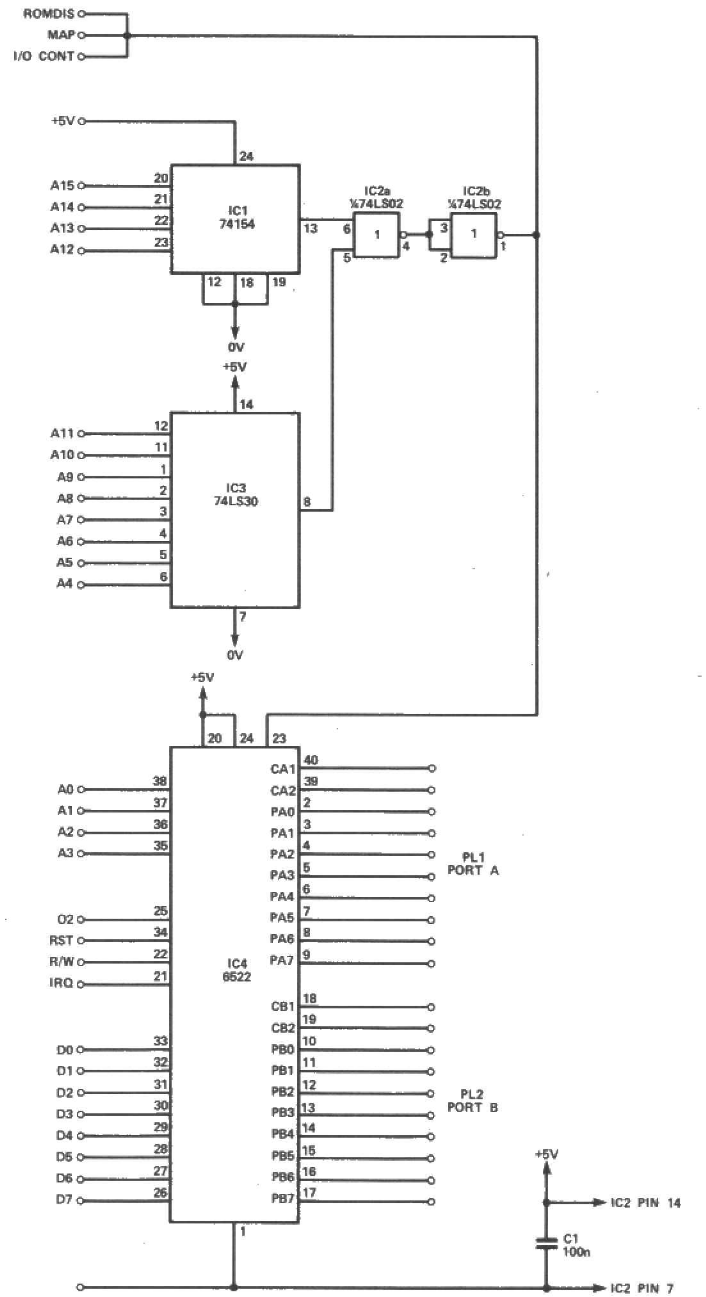


Figure 1. Full circuit diagram of the 6522 input/output interface.

The Circuit

Apart from the 6522 itself the only other circuitry required is the address decoding. **Fig 1** shows the full circuit diagram of the interface.

With the 16K version of the Oric there would presumably be some 32K of memory locations free of RAM or ROM, but the 48K version of the machine has the full 64K address range of the 6502 occupied by RAM (16K of which is overlaid by ROM). However, there are spare memory locations from address #BFE0 to #BFFF (the # sign being used to indicate a hexadecimal number in Oric BASIC), as shown in the memory map (Appendix A) in the Oric manual. The point that has to be kept in mind is that although these addresses are normally unused by the machine, there are nevertheless RAM and possibly ROM located here.

When using the expansion port of the Oric as an output this does not matter since the microprocessor will simply write to both the RAM and the output device. When information is being read from the port the situation is different, as there would be two devices placing an output on the data bus if both the RAM and input circuit were read simultaneously.

The Oric manual does not specify how the internal input/output devices can be cut off from the data bus, but the list of terminals (Appendix F) in the manual does show one marked I/O control. This appears to be an input which is normally high, and taking this pin low when inputting data appeared to be partially effective, but still left some corruption of the data. Experimentally taking the ROMDIS and MAP pins low as well seemed to entirely clear the problem, and allowed an unhindered flow of data onto the data bus.

In the circuit of **Fig 1** IC1 is a 4 to 16 line decoder, and the only output which is used in this case is output 11 (B in hex). This is used to decode the upper four address lines. IC3 is an eight input NAND gate which is used to decode the middle eight address lines, and will only produce a low output if all eight inputs are high (FF in hex). Two NOR gates of IC2 are used to produce a low output only when the outputs of IC1 and IC3 are both low, which only occurs when addresses from #BFF0 to #BFFF are present on the address bus. The output from IC2b is used to drive the negative chip select input of the 6522 (IC4), and the positive chip select input is simply wired to the positive supply rail. The 6522 has sixteen internal registers which are selected using the register select inputs. These are fed from the lower four address lines and the 6522 therefore occupies all the address locations from #BFF0 to #BFFF. There is effectively full address decoding so that the addresses from #BFE0 to #BFEF are free for use with other interfaces.

The eight line data bus connects direct to the bidirectional data bus of the 6522. A 74LS245 transceiver was originally used as a buffer here, but it seemed to be superfluous and has been omitted from the final design. The clock input of the 6522 is fed from the clock (02) terminal of the expansion port, and an important point to note here is that the clock frequency is just 1MHz even though the Oric uses the 1.5MHz 6502A version of the 6502. It is thus unnecessary to use the high speed 6522A device for IC4, and the standard 6522 is perfectly satisfactory.

The reset, read/write, and IRQ (interrupt request) terminals connect to the corresponding lines of the Oric expansion port. The 6522 is therefore reset at switch-on and when the break switch is operated.

The circuit is powered from the Oric power supply via the two output terminals of the expansion port. The power supply seems well able to take the extra load in the short and medium term, but it might be advisable to supply the interface from its own power supply if it is likely to be used for prolonged periods. The supply current required is about 100 milliamps or so. C1 is merely a supply decoupling capacitor.

Note that two gates of IC2 are unused and that no connections are made to the inputs or outputs of either of these.

Construction

Units of this type tend to be a little awkward from the constructional point of view, but the single-sided printed circuit board of **Fig 2** is reasonably straightforward to build and there is little risk of errors being made when using this method of construction. The board will probably be most easy to build if the link wires are added before the integrated circuits and C1 are soldered in place. On the prototype all four integrated circuits are mounted in sockets, and it is strongly recommended that at least IC4 should be fitted in one.

The connection from the board to the Oric is made by way of a 34 way ribbon cable of up to about one metre in length fitted with a 34 way header socker, and ready-made cables of this type are easily obtained. The header socket plugs into the expansion port of the Oric, while the free end of the cable connects to Verpins fitted at the appropriate points on the printed circuit board. Connecting the cable is quite easy provided you strip the insulation from the end of each wire and tin the exposed wire, plus the veropins, with a generous amount of solder. Be careful not to cross over any of the leads and to connect the cable the right way round (refer to **Fig 2** and Appendix F of the Oric manual).

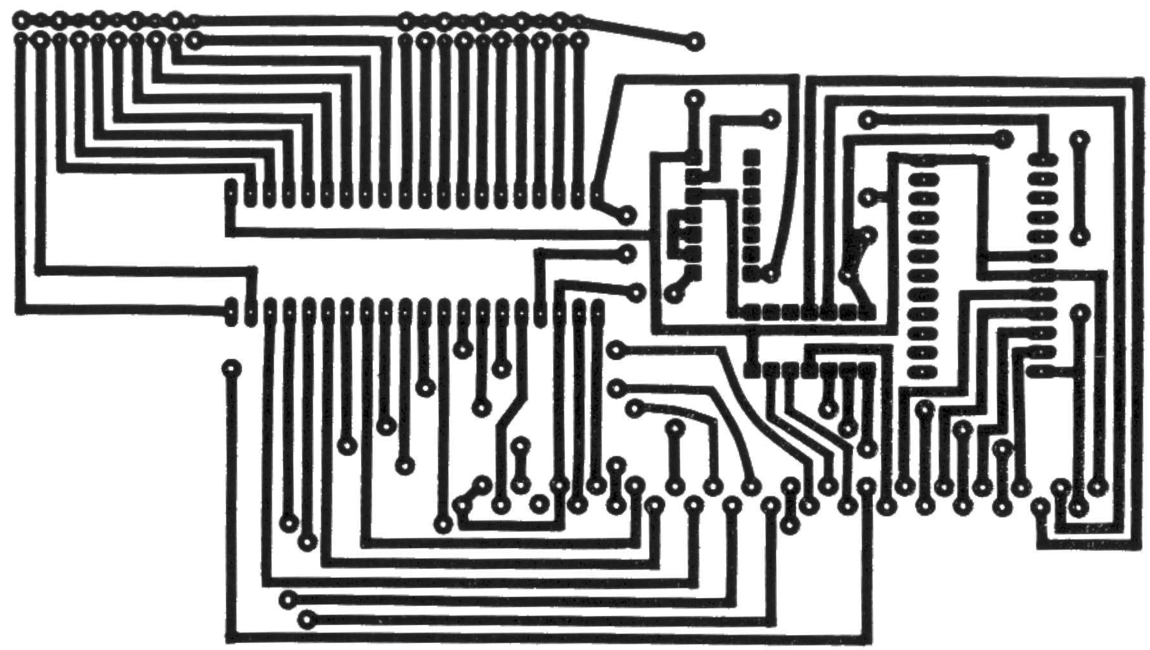
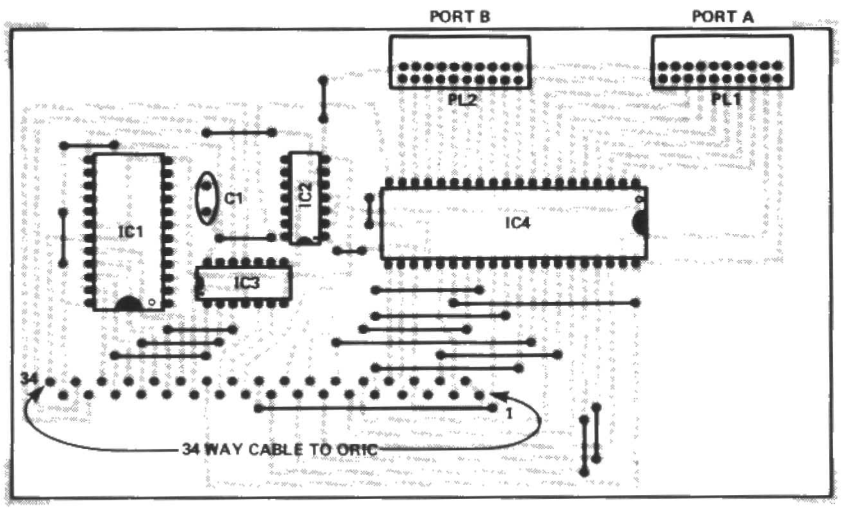


Figure 2. (right) shows the foil pattern of the PCB. This has been designed as a single sided board in order to keep costs down.

The input/output connections of the 6522 are taken to two 20 way IDC plugs, and in each case ten pins are used as ground connections and screens while the other ten carry the eight input/output connections plus the two handshake lines. The pin configurations of the two connectors are slightly different to keep the printed circuit board as uncomplicated as possible) and the two configurations are shown in **Fig 3.** The ports are designated Port A and Port B to correspond to the method of identification used in the 6522 data sheet.

Figure 4. The overlay of the project. It is strongly recommended that IC4 is mounted in a socket. There are a number of links on the board and these are best formed before any other components are fitted.



Testing

Connect the interface to the expansion port of the Oric before switching on, so that the 6522 is reset at switch on. A quick initial check of the unit can be made by typing into the computer:-

PRINT PEEK (#BFF0)

and hitting return. This should give the answer 255 since both ports act as inputs after reset, and both have pull up resistors on all eight data inputs. Port B is a #BFF0 incidentally. If the unit is not operating properly some other answer will be obtained, and this would probably by 85 which is the number normally programmed into #BFF0 and other unused memory locations of the Oric after switch-on. If the right answer is not obtained switch off at once and check the unit thoroughly until the fault is found and rectified.

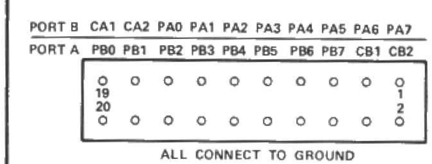


Figure 3. The pin configurations of the two PCB connectors.

converters etc., but the 6522 data sheet should be consulted if there is any doubt as to whether or not a particular device will directly interface to the unit. The data sheet gives a mass of information about voltage levels, drive currents, etc.

Handshaking

Some applications will need no more than the eight input/output lines of each port, but it will often be necessary to additionally use one or both of the handshake lines. The form which the handshaking takes will vary from one application to another and in its most simple form it is just a matter of producing an output pulse to indicate to the peripheral device that new data is available, or of acting on an input pulse which signals to the computer that new data is present at the input. A more sophisticated technique may be required in some cases where, for example, it might be necessary for a peripheral device to indicate that it is ready to receive data, with the computer then sending data to the port and indicating that it has done so. This requires two handshake lines, one acting as an input and the other operating as an output.

The 6522 can handle any normal handshaking requirements, and the handshake lines are not difficult to master.

The handshake lines are made to operate in the required mode by writing the appropriate number to the peripheral control register at BFFC. When a handshake line is used as an input, a bit of the interrupt flag register at BFFD shows whether or not the input has been activated.

If we consider the peripheral control register first, bits 0 to 3 control the Port A handshake lines while bits 4 to 7 function in exactly the same way but for Port B. In order to illustrate the way in which the handshake lines are controlled we will only consider the use of Port A together with bits 0 to 3 of the peripheral control register.

CA1 can only be used as in input and it has just two modes of operation. If bit 0 of the peripheral control register is set at 0 a high to low transition of CA1 sets its interrupt flag, while a 1 in bit 0 of the control register results in the flag being set on a low to high transition of CA1. There are two ways of resetting the interrupt flag to zero, and one of these is to simply to perform a read or write operation to Port A at BFF1. The other is to write a 1 to the appropriate bit of the interrupt flag register (it cannot be set at 0 by writing 0 to this bit of the flag register).

CA2 can be used in four input and four output modes, and it is controlled by bits 1 to 3 of the peripheral control register. **Table 2** shows the available modes and the binary codes/decimal numbers that produce them.

Control Register	Mode of Operation
000/0	Handshake Input Mode (high/low)
001/2	Independent Input Mode (high/low)
010/4	Handshake Input Mode (low/high)
011/6	Independent Input Mode (low/high)
100/8	Handshake Output Mode (high/low)
101/10	Pulse Output Mode (high/low)
110/12	Low Output Mode
111/14	High Output Mode

Table 2. The available modes of CA2 together with the binary codes to produce them.

The handshake input modes are the same modes that CA1 uses. The independent mode is different in that read and write operations to Port A do not reset the interrupt flag, and the only way of doing this is to write a 1 to the flag. In the handshake output mode CA1 is taken low by a write operation to Port A at BFF0, and is reset to the high state by an input to CA1. In the pulse mode it goes low for one clock cycle (about 1 us) when data is sent to Port A at BFF0. In the low and high output modes CA2 simply stays continuously in the specified state.

Thus, to set CA2 in the handshake output mode and CA1 in the low to high mode the number written to the peripheral control register would be 9 (8 + 1) using the command:-

```
POKE #BFFC,9
```

Of course, this sets the control register for Port B at zero, and if this port was in use as well the number POKEd would have to be calculated to set the upper four bits in the correct states as well.

Four bits of the interrupt flag register are used in conjuction with the handshake lines, as detailed below.

Bit 0 (1 in decimal) CA2
Bit 1 (2 in decimal) CA1
Bit 3 (8 in decimal) CB2
Bit 4 (16 in decimal) CB1

In order to check if just one bit of the register has been set the logic AND function is used. For example, PRINT PEEK (-(#BFFD) AND 8 would give an answer of 8 if bit 3 of the interrupt flag register is set, or 0 if it is not. In other words the other bits are masked off and do not affect the number returned. This system can be used with any of the other bits of the register, but be careful to AND the PEEKed value with the correct number.

Apart from the two ports and the two timer/counters the 6522 also has a serial register, and input latches on both ports. There is not enough space to go into greater detail here, and anyone using the interface would be well advised to obtain and carefully study the 6522 data sheet.

PARTS LIST

Capacitor

C1	100n

Semiconductors

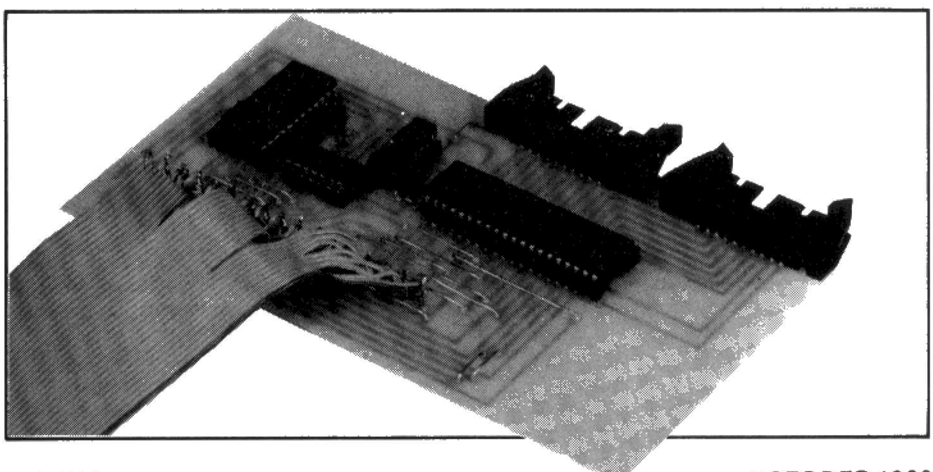
IC1	74LS154 or 74154
IC2	74LS02
IC3	74LS30
IC4	6522

Connectors

PL1, 2	20 way IDC plugs

Miscellaneous

PCB, Veropins, 34 way ribbon cable fitted with 34 way IDC header socket, DIL sockets, Wire, etc.



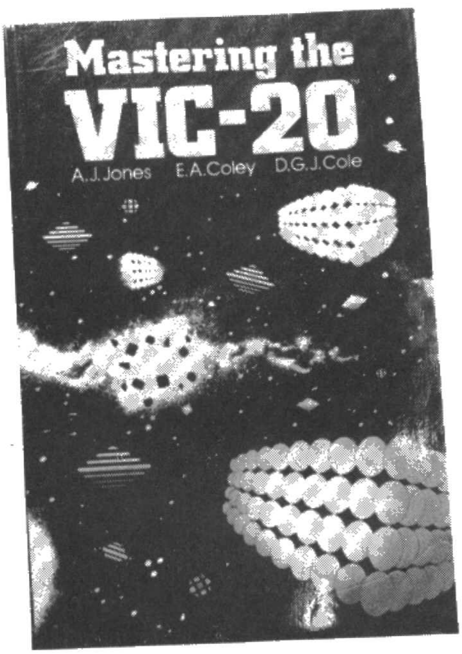
Each month Harry Fairhead will look at a selection of recently published books and software.

The vast array of books and software currently available presents the microcomputer user with a bewildering choice. This page is intended to help readers by presenting an informed and considered opinion about a couple of books and pieces of software each month. Because space is limited, this review page will be selective, and only books and software that seem to offer a good deal will be included. Every book reviewed on this page can be obtained through the ECM Book Service.

RECOMMENDED READING

Mastering the VIC-20
by A. J. Jones, E. A. Coley and D. G. J. Cole
Ellis Horwood Limited, 1983.

The VIC-20 is a popular micro that has been rather neglected when it comes to books. Its users will therefore be pleased to welcome a book specifically about their machine. That this is not a book for the beginner is apparent in the first chapter, which presents a breathtakingly whirlwind tour through the topics of variables, strings, arrays, FOR loops,

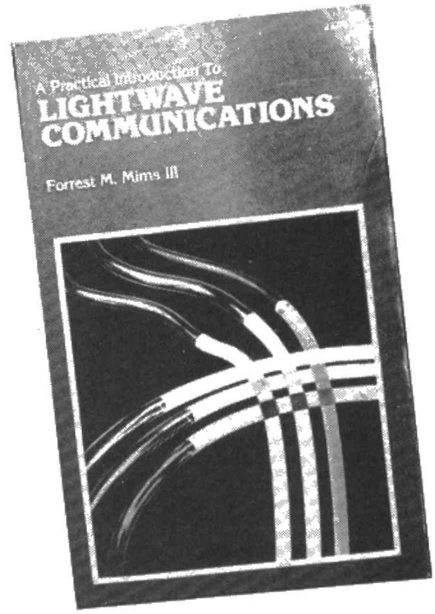


structured programming, PEEK, POKE and binary arithmetic and, finally, logical operators. The reader needs to have a good grasp of all this material before they read the chapter so little is done to explain the ideas introduced. The BASIC program contained in this chapter enables the VIC to be used as a two octave keyboard. It is therefore a particularly attractive program and one that uses the techniques discussed, although the link with the rest of the chapter is perhaps not clarified sufficiently. The second chapter tackles the problem of array storage and introduces the idea of using the cassette buffer memory to store a byte array. In the next chapter an alternative way of acquiring space for this purpose, by lowering the top of memory, is presented along with an introduction to the 6561 video generator's control registers. Chapter Four deals with user-defined graphics and hi-res graphics and again covers a lot of ground very rapidly. The next two chapters are about peripherals – the keyboard, the cassette unit, the disk drive and IEEE-488 interface, and the printer – and accessories – the games port, memory expansion port and user port. Chapter Seven is entitled "System Architecture" and presents a block diagram, a discussion of interrupts, the BASIC interpreter and the 'Kernal' machine code subroutines which can be accessed from the users own machine code programs. The final chapter is a comprehensive, and therefore necessarily compressed, introduction to machine code programming. Two appendices are devoted to programs. Appendix I presents a BASIC program for a 'Startrek-type' game and Appendix II gives two hi-res graphics routines for drawing and saving and for printing them out. The final appendix is devoted to the memory map, ASCII codes and keyword codes – all useful data.

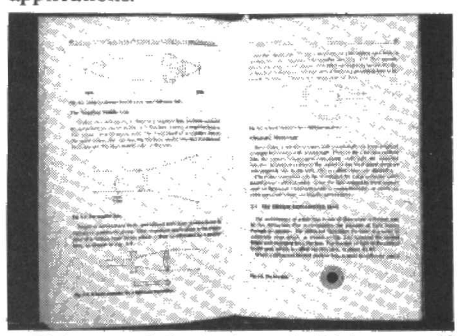
This book has been written by professionals who are used to much larger computer systems than a humble micro. This certainly shows in their approach and style and limits the book's appeal. If you are a proficient programmer and already understand your VIC you will find a lot of useful information in this book. If you are a raw beginner then it will not help you to master the rudimentary aspects of computing.

A Practical Introduction to Lightwave Communication
by Forrest M. Mims III
Howard W. Sams, 1982

Fibre optics and their use in communications systems is a subject that is much talked about but, apart from very costly professional systems, very little is actually done. In part this lack of action is due to the difficulty in finding the right sort of information. Books either seem to deal with the Heath Robinson approach using garden hose pipes filled with water or plastic filaments obtained from clothes lines, or are so academic that it is difficult to find any English between the



formulae! I am happy to say that "A Practical Introduction to Lightwave Communication" falls into neither of these categories. Starting off with a little history, the book quickly moves on to consider the simple optical theory behind the use of lenses as collimators and the way that light travels in fibre light guides. The treatment is at about the right level to introduce such topics to the electronic engineer. Chapter Four outlines the problems encountered in handling fibres, cutting and joining etc. Chapter Five deals with the electronic side of the technique, that is, modulation, transmission, coding etc. The characteristics of entire systems including loss and noise analysis are investigated. Although fibre optics must be the most interesting area of lightwave communications from the point of view of computers, free space transmission is also covered and this might provide a few ideas for novel applications.

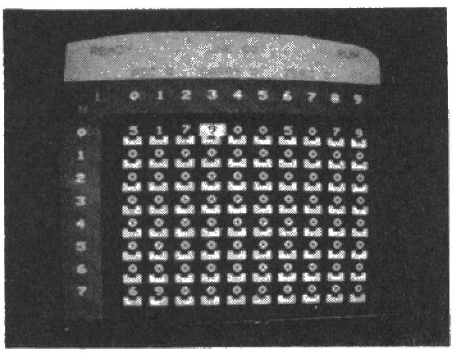


This certainly isn't a build-it-yourself book. However it does outline what you need to know to evaluate and design your own fibre optic systems. This knowledge coupled with one of the many fibre optic evaluation kits that are now on the market (see RS components for example) should produce something of practical value. There is still the need for something a little more directly aimed at the small computer user and experimenter but for anyone with a background in electronics, lightwave communication is a book that will provide much information.

SELECTED SOFTWARE

Peeko-Computer
for the BBC Micro Models A and B
Acornsoft

Peeko-computer is not a new idea, it first saw the light of day running on the Acorn Atom, but it is a program that deserves to be better known. It is a software simulation of a simple computer that, not surprisingly, looks very like a simplified 6502 microprocessor. It has a single accumulator, or A register, and 80 bytes of memory. The instruction set is also considerably reduced to just 10 instructions, although you can load an additional set of 10 when you are ready to try something more advanced. The standard 10 instructions include load and store, addition and conditional jump so you can write some fairly convincing short demonstration programs.



The advantage of using a simulation is that the program can be executed an instruction at a time and the contents of all the memory locations and the register are displayed at each step. Above all a simulation is as safe and friendly as BASIC and complete beginners can enter programs and make mistakes without losing control of the machine and, along with it, any hope of finding their mistake! Peeko computer is a good machine code trainer and should at least be considered by anyone about to learn or teach machine code, especially 6502 machine code.

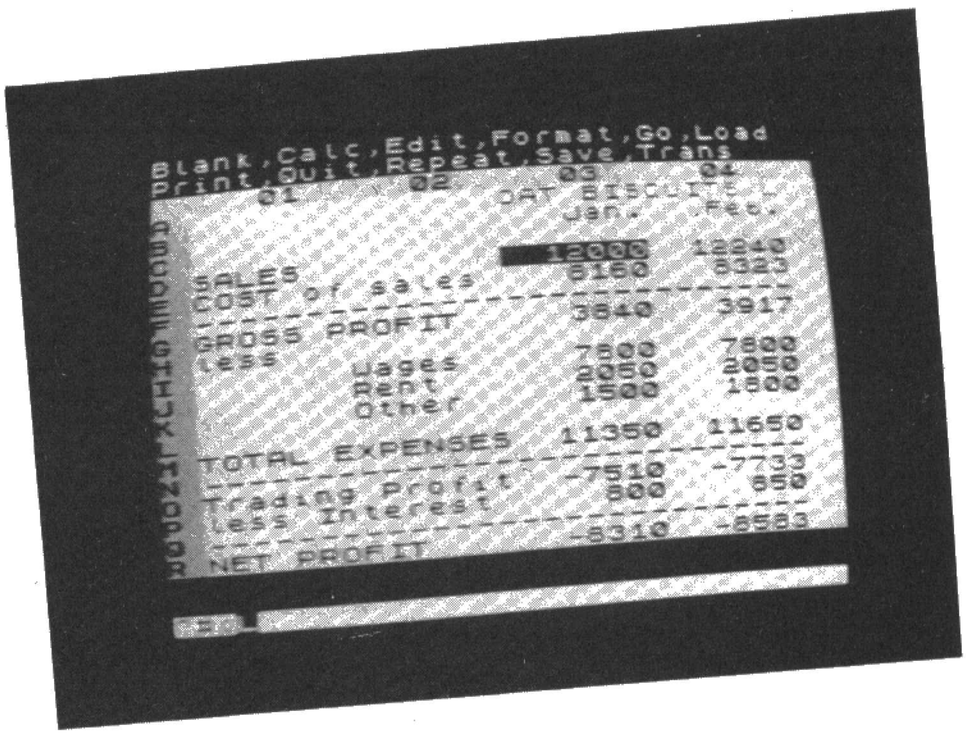
Vu-Calc
for the ZX Spectrum 16K or 48K
Psion

Spread sheet calculators are one of the most-sought-after programs for home computers. Essentially they allow you to handle data in rows and columns and apply mathematical formulae in an ordered way. They are therefore the tool you need for financial or simple statistical analysis. There is an initial problem to overcome with any spread sheet calculator and getting to know how to use it is difficult – especially if you are not used to handling data ina similar way by old fashioned methods. However, if you have a need for this type of analysis – either for a small business or equally for household accounts – it is well worth the effort of putting in the time needed to let your computer do the number crunching for you. It saves time later in trying to balance the books by finding out where your mental arithmetic went wrong!

The only documentation with this program is contained on the cassette inlay which folds out to give seven pages of information. This gives the details you require but if you are already familiar with spread sheet programs in general it does not give you a feel for the software's potential. Fortunately Psion have included an example on the reverse side of the tape which you can tinker with to learn about its features. I rather wish that the example had been discussed in the program notes, perhaps even with exercises – after all many Spectrum users will never have used similar programs.

Vu-Calc is reasonably fast at applying formulae by virtue of being written almost entirely in machine code. The grid used by Vu-Calc is larger than the screen area and so the screen acts as a 'window' and you see part of the whole spread sheet at any one time. Using the cursor keys you can move your position on the screen and the window shifts, revealing different sections of the whole grid in response to the cursor's movement. It has a fairly simple menu structure and you have four options for formatting data.

Although it has been written for either a 16K or a 48K Spectrum it is easier to use with the larger RAM capacity. With 48K you have more room for formulae and this is very important if you are trying to do analysis of any complexity. There is an editing facility for altering formulae which is most useful if you want to do more than one type of analysis on the same date. This is the sort of program you need to use in order to discover its potential and you need practice with it before you can use it successfully. Personally I've got lots of jobs lined up for it that I've been laboriously doing by hand and I'm very glad to have made its acquaintance. **E&CM**



SPEECH RECOGNITION SYSTEMS

Mike Furminger, of Nene College, Northampton, outlines the basic principles of a micro based speech recognition unit.

The idea of talking to a computer then getting an answer in plain English has long been a popular one in science fiction. The reality of talking to a computer though is not quite as easy as having a conversation with another person. Speech Recognition is being used in modern helicopters and commercial planes, however, and enables pilots to call up desired displays on a VDU rather than looking for an instrument amongst many required by the aircraft.

Alternative Approaches

Computers can understand certain words when an operator has 'taught' the computer the word before or, can analyse each syllable of the sound and try to create a reasonable match to the that word against a stored library of sounds. It is possible to buy speech recognition boards which will recognise up to 100 words with 99% reliability, however these are expensive and unsuitable for home use.

To make a small home computer recognise words is a little tricky, but reasonable results can be obtained if the computer is not expected to discriminate between similar words spoken by the same person. In order to appreciate how machines can understand the spoken word it's necessary to examine the basics of human speech.

In principle spoken words are made up of a series of basic elements. There are 'Voiced' and 'Unvoiced' sounds, i.e. sound which uses the mouth to form the noise and sound which is just blown from the lungs. An example of unvoiced sound is the word 'Oh'. The voiced sounds are very complex but simply can be identified as one of three major groups i.e. fricative, sibilant and plosive sound. Fricative sound uses the tongue to make the sound as in 'fur', sibilant sound is the 's' sound and plosive sound is a sudden release of air by the mouth as in the word 'pop'. Although the above is not a complete analysis of spoken sound it is adequate for an introduction.

Analysis Techniques

To analyse these different sounds so that a computer can recognise different words it's necessary to use some technique which breaks down the complex wave form into a manageable spectrum. If we had a large mainframe computer to hand then we could try, for example, the linear predictive coding (LPC) technique used by the Texas company's speech chips. In principle LPC creates a series of speech frames (eg 40 per second) in which the frequency, amplitude, duration and sound type is converted to a binary code. This binary code is then saved in a relatively small memory. Recognition would occur if an incoming speech pattern matched a known one.

The techniques of LPC are beyond the scope of a small microcomputer but simpler approaches to the problem can be used. A spectrum analyser, such as used to control disco lights, would be a suitable method. The spectrum analyser breaks down a complex spoken word or musical note into its frequency elements. This is similar to the way in which a prism breaks white light into different colours but instead a sound is broken down to elements like notes on a piano keyboard.

Spectrum Analysers

Many designs for spectrum analysers have been published by various magazines. These are either based on a series of op-amp filters or use

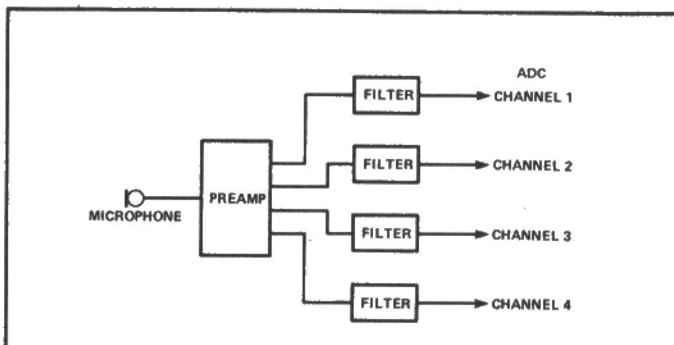
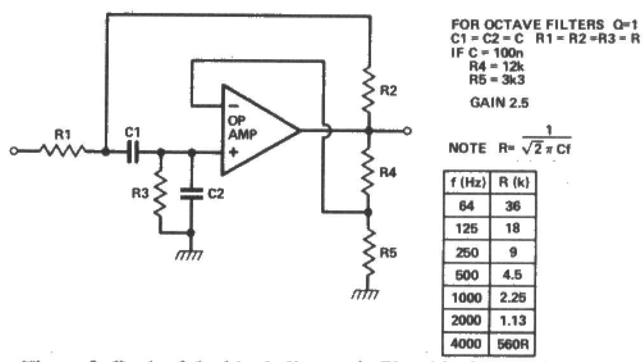


Figure 1. Block diagram of a Spectrum Analyser. After amplification, the speech signal is divided into four frequency bands.



f (Hz)	R (k)
64	36
125	18
250	9
500	4.5
1000	2.25
2000	1.13
4000	560R

Figure 2. Each of the block diagram's filter blocks could take the form of an active filter based around an op-amp.

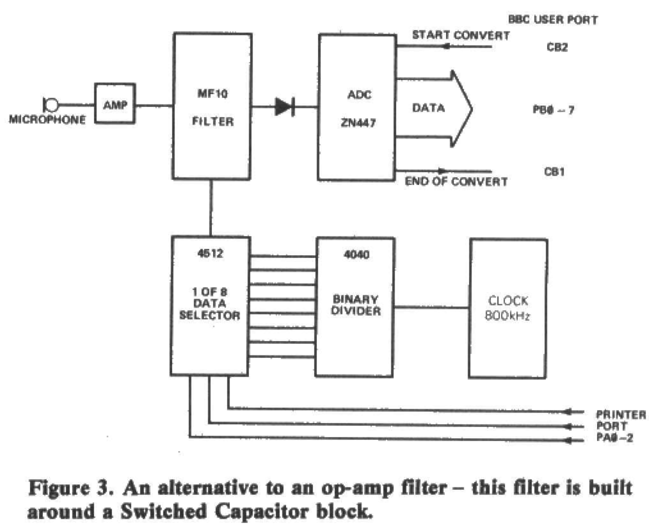


Figure 3. An alternative to an op-amp filter – this filter is built around a Switched Capacitor block.

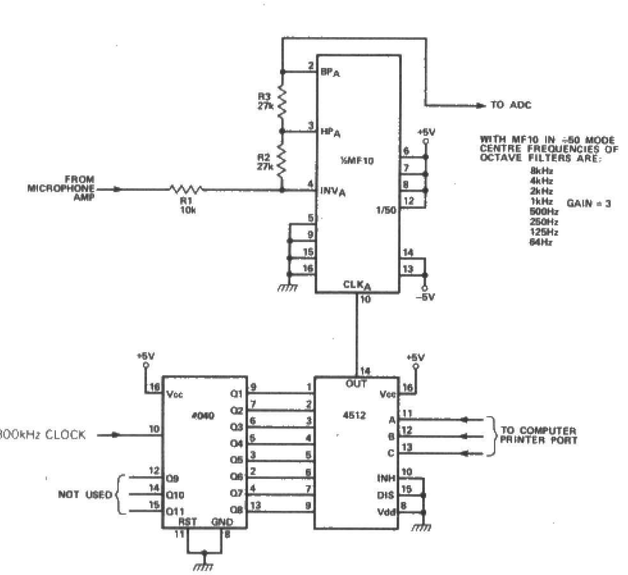


Figure 4. Circuit details of the filter section of the block diagram of Fig 3.

Program 1. This illustrates the way in which words can be saved as in the outline of Fig 5a.

```
100 PRINT"HAVE YOU TURNED ON THE"
110 PRINT"REAL TIME SPECTRUM ANALYZER?"
120 GETA$:IFA$<>"Y"THEN120
130 DIMA(30),B(30),C(30),D(30),E(30),F(10)
140 SYS4096*11
150 PRINT"THIS PROGRAM TAKES THREE COPIES OF"
160 PRINT"ONE WORD AND SAVES IT ON DISC"
170 INPUT"ENTER WORD AND PRESS RETURN",W$
180 FORM=1TO3
190 POKE46080,1 REM GAIN MAX
200 A=USR(5) POKE931,255 A=USR(3) REM CLEAR
210 A=USR(7) A=USR(8)
220 A=USR(4)
230 POKE931,0 REM FREEZE
240 PRINT"WHEN YOU ARE READY FOR TEST ";M;" PRESS SPACE AND SAY ",W$
250 GETA$:IFA$<>CHR$(32)THEN250
260 FORI=0TO300:NEXT
270 A=USR(1)
280 FORK=1TO50
290 A=USR(3):REM SCAN FILTERS
300 A=USR(2)
310 NEXT
320 PRINT"SORTING DATA"

READY.
READY.

330 FORF=0 29 REM FILL ARRAY
340 A(F)=PEEK( 27+F)
350 NEXT
360 KK=3:REM FIND PEEKS FROM 50HZ
370 FORII=0TO12
380 KK=KK+1
390 IFKK>28THEN480
400 IFA(KK)>A(KK+1)THEN420
410 IFA(KK)<=A(KK+1)THEN380
420 B(II)=KK+1 C(II)=A(KK)
430 KK=KK+1
440 IFKK>28THEN480
450 IFA(KK)<=A(KK+1)THEN470
460 GOTO430
470 NEXTII
480 FORI=0TO12:SW=0:REM BUBBLE SORT
490 FORJ=0TO12
500 IFC(J)>=C(J+1)THEN550
510 T=C(J) C(J)=C(J+1):C(J+1)=T SW=1
520 T=B(J) B(J)=B(J+1):B(J+1)=T
550 NEXTJ
560 IFSW=0THENI=12
570 NEXTI
575 IFM=1THENFORI=0TO12:D(I)=B(I):NEXT
576 IFM=2THENFORI=0TO12:E(I)=B(I):NEXT
580 NEXTM
590 FORS=0TO9:REM COMPARE
600 F(S)=B(S)
610 IFD(S)=E(S)THENF(S)=D(S)
620 IFF(S)=0THENF(S)=B(S)
630 NEXTS
640 FL$="@0:"+W$+",SEQ,WRITE"
650 OPEN1,8,15:OPEN2,8,14:CLOSE2
660 OPEN2,8,14,FL$
670 FORW=0TO9:PRINT#2,F(W);CHR$(13);:NEXT
680 CLOSE2:CLOSE1
690 PRINT"ANOTHER WORD?"
700 GETA$:IFA$="Y"THEN170
710 IFA$="N"THENEND
720 GOTO700
READY.
```

a switched capacitor filter such as tne MF10. Either approach will work well enough, so it a spectrum analyser is already to hand then this is another use for it. If you have not got a spectrum analyser then building a basic model is not too difficult. With a BBC model B computer a very simple device could be built using filters into each of the 4 ADC channels. Suitable filter frequencies are mentioned later. All other computers will require a multiplexed ADC to accept the signal data.

Another approach is to use an MF10 controlled with various clock rates to reproduce many filters in one as shown in **Fig. 3.**

Sound Experiments

The author used a PET computer fitted with an Eventide 30 1/3 octave filter spectrum analyser and consequently all examples are quoted for a PET computer, the method described though can be transferred to any other machine. The properties of speed will dictate which filters are significant.

Unvoiced sound tends to have a broad spectrum with a maximum around 500 Hz, the exact frequency depending on the physical size of the speaker. Voiced sounds can be identified by their properties, since sibilant sound is a high frequency hiss around 8 KHz, plosives are low frequency sounds around 150 Hz and fricative sound tends to be more difficult to identify but mostly its components are around 1 KHz. There is no useful information to be found below 50 KHz or above 10 KHz. If only a few channels are used then it is a good idea to keep the above points in mind.

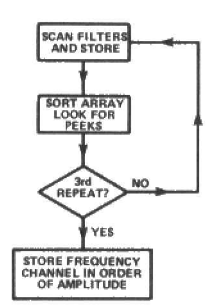
A suitable 8 channel octave set of filters would have centre frequencies at 62, 125, 250, 500, 1000, 2000, 4000 and 8000 Hz.

Software Outlines

The software for speech recognition on a microcomputer would first store a copy of a word, then compare any incoming word with the stored copy. The more comparisons that are required the slower will be the response.

The program elements for saving a copy of a word are shown in **Fig 5a.**

An example of these procedures on the PET computer are given in **program 1,** lines 330 to 570. The program fills an array with the stored data from the spectrum analyser, sorts for peaks and then sorts again into channel order with a bubble sort.



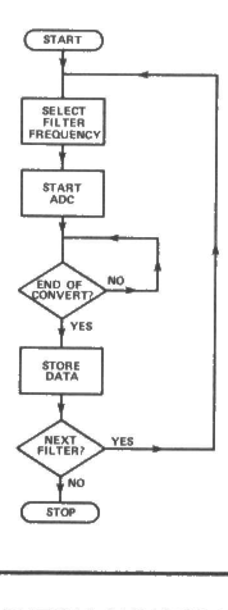
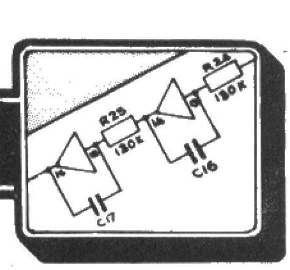


Figure 5a (above) shows the program elements for saving a copy of a word. Fig 5b (right) shows the flow diagram of a Spectrum Analyser based on an MF10 filter.



Program 2. This illustrates the way in which a computer could perform a simple 'sex test' based on the fact that males have peak vocal frequencies of less than 500 Hz while females have higher dominant frequencies.

```
100 PRINT"HAVE YOU TURNED ON THE"
110 PRINT"REAL TIME SPECTRUM ANALYZER?"
120 GETA$:IFA$<>"Y"THEN120
125 PRINT"COLLECTING DATA"
130 DIMA(30),B(30),C(30)
140 SYS4096*11
150 K=1:W$="UP":GOSUB700
160 K=2:W$="DOWN":GOSUB700
170 K=3:W$="LEFT":GOSUB700
180 K=4:W$="RIGHT":GOSUB700
190 PRINT"THE COMPUTER IS NOW READY TO PLAY"
210 FORI=1TO2000:NEXT:GOTO1100
220 POKE46080,1:REM GAIN MAX
230 A=USR(5):POKE931,255:A=USR(3):REM CLEAR
240 A=USR(7):A=USR(8)
250 A=USR(4)
260 POKE931,0:REM FREEZE
280 PRINT"YES?"
285 A=USR(3):IF PEEK(826)<50THEN285
290 FORK=1TO100
300 A=USR(3)
310 NEXT
320 PRINT"     "
330 FORF=0TO29STEP2
340 A(F)=PEEK(827+F)+PEEK(828+F)+PEEK(829+F)
350 NEXT
360 KK=0:REM FIND PEEKS FROM 50HZ
370 FORII=0TO12
380 KK=KK+1
390 IFKK>28THEN480
400 IFA(KK)>A(KK+1)THEN420
410 IFA(KK)<=A(KK+1)THEN380
420 B(II)=KK+1:C(II)=A(KK)
430 KK=KK+1
440 IFKK>28THEN480
450 IFA(KK)<=A(KK+1)THEN470
460 GOTO430
470 NEXTII
480 FORI=0TO12:SW=0:REM BUBBLE SORT
490 FORJ=0TO12
500 IFC(J)>=C(J+1)THEN530
510 T=C(J):C(J)=C(J+1):C(J+1)=T:SW=1
520 T=B(J):B(J)=B(J+1):B(J+1)=T
530 NEXTJ
540 IFSW=0THENI=12
550 NEXTI
570 R0=0:R1=0:R2=0:R3=0
580 FORI=0TO9
590 IFH0(I)=B(I)THENR0=R0+1
600 IFH1(I)=B(I)THENR1=R1+1
610 IFH2(I)=B(I)THENR2=R2+1
620 IFH3(I)=B(I)THENR3=R3+1
630 NEXTI
640 IFR0>=R1ANDR0>=R2ANDR0>=R3THENP=P+40:RETURN
650 IFR1>=R0ANDR1>=R2ANDR1>=R3THENP=P-40:RETURN
660 IFR2>=R0ANDR2>=R1ANDR2>=R3THENP=P-1:RETURN
670 IFR3>=R0ANDR3>=R1ANDR3>=R2THENP=P+1:RETURN
680 GOTO220
690 END
700 REM COLLECT FILES
710 OPEN1,8,15
720 FL$="0:"+W$+",SEQ,READ"
730 OPEN3,8,13,FL$
740 FORW=0TO9
750 INPUT#3,H(W)
760 ONKGOSUB810,820,830,840
770 NEXTW
780 CLOSE3
790 CLOSE1
800 RETURN
810 H0(W)=H(W):RETURN
820 H1(W)=H(W):RETURN
830 H2(W)=H(W):RETURN
840 H3(W)=H(W):RETURN
850 REM GAME
860 DATA"                                        "
870 DATA"                                        "
880 DATA"                                        "
890 DATA"                                        "
900 DATA"                                        "
910 DATA"                                        "
920 DATA"                                        "
930 DATA"                                        "
940 DATA"      *                                 "
950 DATA"                                        "
960 DATA"                                        "
970 DATA"                                        "
980 DATA"                                        "
990 DATA"                                        "
1000 DATA"                                       "
1010 DATA"                                       "
1020 DATA"                                       "
1030 DATA"                                       "
1040 DATA"                                       "
1050 DATA" RECORD  100      SEC                  "
1060 DATA"                                       "
1070 DATA"                                       "
1080 DATA"                                       "
1090 REMRITE TO SCREEN
1100 PRINT"":PRINT
1110 FORI=1TO20:READP$:PRINTP$;:NEXT
1120 RESTORE
1130 T3=VAL(MID$(P$,8,7)):IFT3<10THENT3=100
1140 P=33173:POKEP,42
1170 T=TI
1180 P1=P
1190 GOSUB220
1200 IFP<>P1THEN1220
1210 PRINT"STOPPED":GOTO1530
1220 Q=PEEK(P)
1230 IFQ=102THEN1270
1240 IFQ=97THEN1390
1250 POKEP1,32:POKEP,42
1260 GOTO1180
1270 REMCRASH
1280 FORI=1TO100
1290 POKEP,97
1300 POKEP,105
1310 POKEP,95
1320 POKEP,98
1330 POKEP,105
1340 POKEP,95
1350 NEXT
1360 POKEP,42
1370 POKEP1,32
1380 PRINT"";:PRINT"CRASHED                          ":GOTO1530
1390 T2=TI:T1=(T2-T)/60:T1=INT(100*T1):T1=T1/100
1400 PRINT"LAP TIME"T1"SEC
1410 IFT1<T3THENPRINT"NEW RECORD"
1420 IFT1<T3THENGOSUB1440
1430 GOTO1530
1440 REMSTORE RECORD
1450 S=1952
1460 T$=STR$(T1):L=LEN(T$)
1470 FORI=1TOL
1480 Y$=MID$(T$,I,1)
1490 Y=ASC(Y$)
1500 POKE(S+I),Y
1510 NEXT
1520 RETURN
1530 FORW=1TO1000:NEXT
1540 GOTO1090
READY.
```

It is a good idea to loop round the recording program 3 times and take the best match. This ensures any odd noises are removed, as in lines 575-630.

A simple 'sex test' could now be performed since the maximum energy channel would indicate the frequency which is significant to the speaker. Males have peak frequencies of less than 500 Hz, females and children have a higher dominant frequency. This is not foolproof but good fun.

Speech recognition with this system is possible if the order of channels is saved in a file. Then when an incoming word is required to be recognised the stored files are compared with the channel order of the incoming word. This can be a 50% match being OK or finding the best match. Neither system is particularly good in the simple form. The problem is that BASIC is slow and if several matches are made for one word (about 16 per second would be a good idea) then the sort and compare becomes too slow. A job for machine code!

If only simple words or noises are compared then a single sample will work well. To demonstrate this try out a musical instrument and see if different notes are recognised. A game can be written where words or music drives a car round a track, or the words are used to control a simple robot. This system only allows pre-recorded words to be compared whereas some professional systems attempt to recognise words as they are spoken. The principles of this system are simple. The challenge is for you to improve the system.

E&CM

HI-RES COMPUTER - THE ANALOGUE BOARD

Paul Izod and Alan Stirling describe the means by which the *E&CM* computer can gain access to the world of measurement and control.

FEATURES

- ★ **Choice of inexpensive 8 bit or high performance 12 bit A-D convertor**
- ★ **8 analogue input channels, either single ended or differential.**
- ★ **Provision for sample and hold amplifier.**
- ★ **A-D conversion time of less than 35us per channel (12 bit)**
- ★ **Conversion may be triggered externally.**
- ★ **Minimal S/W overhead once the board is set up.**
- ★ **Up to 8 boards per system for 64 channels.**
- ★ **2 low cost 12 bit D-A convertors.**
- ★ **8 digital O/P lines.**
- ★ **8 digital I/P lines with external clock/enable.**
- ★ **All I/O connections may be from PCB connector block on front of the board.**

The analogue board has been designed with a wide range of users and applications in mind, and for this reason may appear somewhat complex. If all the facilities aren't required then the component count may be substantially reduced.

The digital I/O and analogue output stages are very straightforward, but an overview of the analogue input circuitry may be useful.

A configuration register is used to specify the channel number or scan range, select single or repetitive scan and whether internally or externally triggered. The board will then select the first channel and keep the Sample and Hold (S & H) in sample mode. When the trigger arrives, the S & H holds the input signal, the trigger is synchronised with the processor clock and the analogue to digital conversion started. After the signal is digitised it is stored in the internal RAM, and the next channel is selected (well – almost; a pipe-line is used to speed up the process). This continues until every specified channel has been digitised. The board will then either wait for the next trigger or proceed immediately to the first channel again, according to the configuration register.

Access to the internal RAM is interleaved between the board and the processor using the E clock. The processor may therefore read the RAM at any time.

Bus Interface

D1 and D2 decode address bits A17 down to A8, establishing the board at F2XX within a 64K boundary. Address bit A7 is used to separate the block of RAM addresses from the block of configuration, DAC output registers and byte I/O addresses. The address map is arranged to accommodate up to 8 of these boards, such that the RAM addresses appear continuous, to facilitate rapid reading of the analogue input channels. A6, A5 and A4 are decoded by D3 in conjunction with a 1 out of 8 switch. The switch position determines the board number and hence address within the system. See **Table 1.** The switch common is the board decode signal. This enables the data bus buffer and is used in the remaining decoding circuitry.

D20 is completely enabled only when the board is addressed on a write cycle. The reason the CLK (via D9) appears in the enable logic of D20 both indirectly via D3 and D15, and directly from D9, is to decode A7 low and ensure that D20 is deselected rapidly at the end of a processor cycle, before the data on the bus becomes invalid. This is why ALS parts are specified for D9 and D20. It is necessary to latch data on a write cycle at the tail end of the cycle because the on board RAM is updated whilst CLK (processor E clock) is low.

D28 decodes the Read signals for Input byte and the high and lower order bytes of RAM.

DAC Output Registers

D52 and D53 supply the least and most significant bytes respectively for DAC0; similarly D55 and D56 drive DAC1. It should be noted that the DAC80 requires inverted input data, and as all registers on the data bus are cleared on reset, the outputs of DAC0 and DAC1 will rise to their most positive value.

Configuration Register and Timing Circuitry

Bits D0, D1 and D2 of D10 set the channel number or scan range. If bit D4 is set, then as counter D13 is incremented such that its output equals scan range, as determined by comparator D11, the counter will be synchronously cleared next cycle. If bit D4 is cleared though, the channel number in the configuration register will always be loaded into the counter. Bit 5 is used to enable or inhibit the retriggering of the timing generator internally, via D17. Bit 6 determines whether the trigger will be internal or external. When Bit 6 is high D19 will always be enabled, regardless of the external Capture signal.

Assume that the configuration register has been set up for repeated acquisition of channel 1, to be initiated by external trigger. On the rising edge of Capture (supplied externally) flip-flop D21 will be clocked, transferring 0 to Q and 1 to Q. The Q output controls the Sample and Hold, so this freezes the input signla. The D input of D22 is now high. This and the remainder of the timing circuit are clocked by the E clock ÷ 2, to give a frequency of 500kHz. The output of D22 and then D23 will therefore change in state according to E, thereby synchronising the external trigger to the clock. Two stages are necessary in case the set-up time of D22 wasn't met and its output entered an unstable state. This would very likely have ended before its output was transferred to D23 on the next edge of clock. Both edges of clock are used to expedite this process.

The Q output of D23 clocks both the address counter D13 and the address latch, D14. This enables the latch to hold the address of the signal in the S & H, whilst the address counter is incremented. By selecting the next channel now, the maximum settling time is given to the instrumentation amplifier, before its output is sampled by the S & H.

D24 and D35 produce the Start Conversion signal for the A-D convertor. It is important that this signal is synchronised to the convertor clock. The system now waits for the End of Conversion signal. When this occurs the output of D36 goes low, presetting D31, priming the trigger circuit and switching the S & H back into sample mode. The low from D36 is clocked through D30 which, via D31-D34, generates the RAM enable and write pulses (whilst E is low). The data is now secure in the memory. As the circuit was configured for repetitive conversion, when QD of D30 goes low, it will clock D21 (via D18 & D19) and the whole process will be repeated.

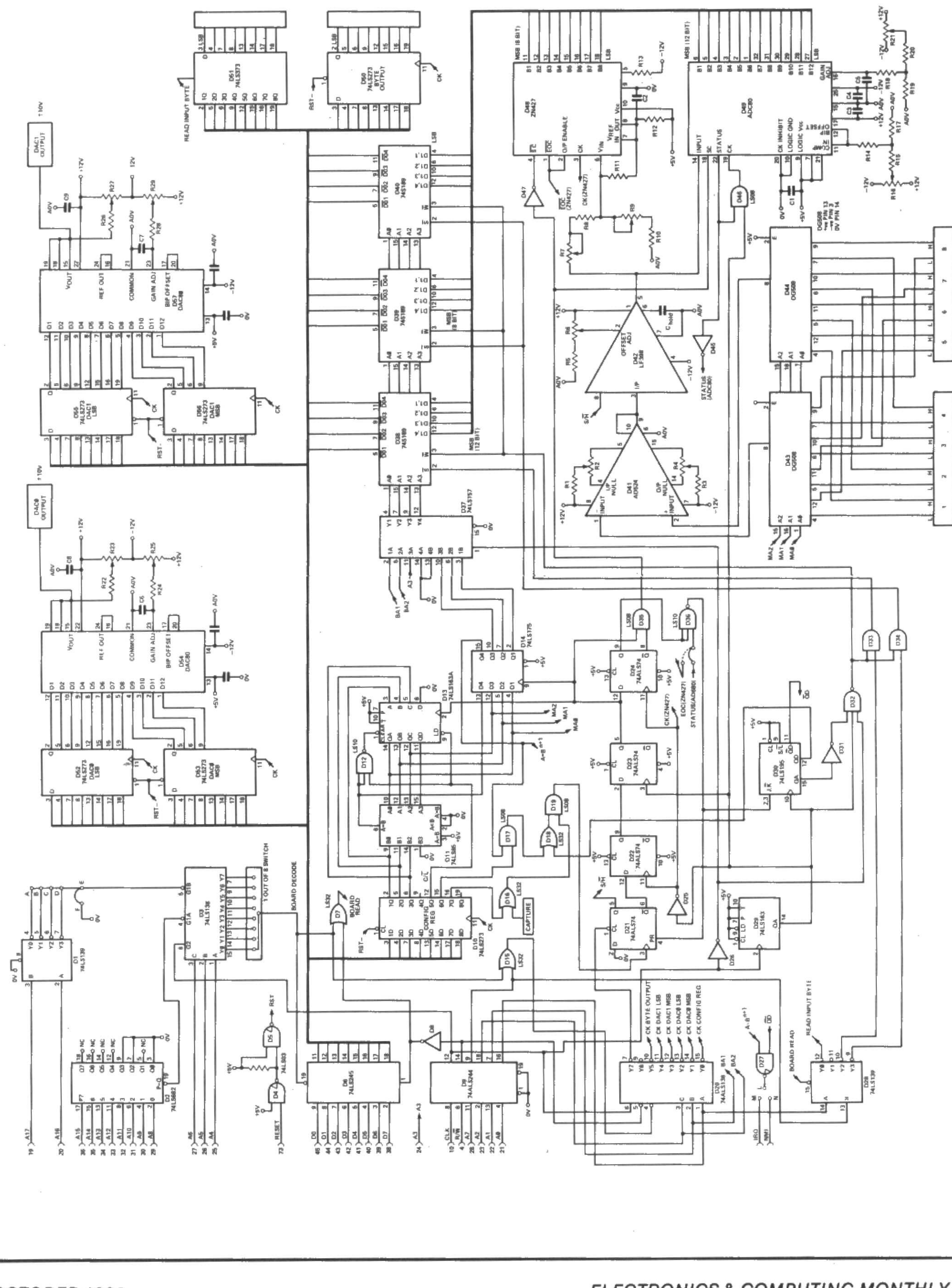


Figure 1. The full circuit diagram of the analogue board.

The Data RAMs

D3 is a 2:1 line multiplexer controlled by the processor E clock. When E is low and the board controls the internal data bus, the RAM address is derived from the address latch D14, and all 3 RAMS are updated together at the end of a conversion. When E is high the RAM address is from address bits A3, A2 and A1. A0 is used to select the upper or lower byte.

Note that the 74S189/74LS189 memories shown invert the data and are therefore suitable for use with the ADL80 (which has complemented outputs). If the ZN427 is used, the non-inverting 74LS219A may be more appropriate.

Power Supplies

The performance of this analogue board depends to a significant extent on the voltage, stability and noise of the power supplies. Although switch mode power supplies are suitable for the +5V, it is unlikely that they would be satisfactory for the analogue circuits, especially if 12 bit performance is required.

Best overall performance will be with ±15V supplies, and this will allow the full range of multiply-sourced devices to be used. However if it is preferred to standardise on ±12V, to maintain compatability with RS232 ports etc., this is possible – provided that the following points are observed:

(a) Maximum analogue input voltage ±9.5V (limited by S & H).

(b) DAC should be either National Semiconductors DAC1280A or equivalent.

(c) ADC either ZN427 or ADC80Z-12, ADC80AGZ-12 or equivalent.

Appendix

Sampling Dynamic Signals

It was briefly mentioned earlier that a Sample and Hold amplifier was necessary when digitising signals whose rate of change of voltage was fast compared to the speed of the convertor.

To illustrate this point, consider a sine wave of amplitude ±10V peak. The maximum rate of change of voltage will be at the zero crossing point. To introduce an error in the conversion of just ½ LSB (least significant bit), the input voltage must change by less than that voltage (½ LSB) during the conversion.

For a convertor of range ±10V and 12 bits of resolution, the voltage presented by ½ LSB is:

$$\tfrac{1}{2} \times \frac{20V}{2^{12}} = \frac{10}{4096}\ V = 2.44mV$$

The instantaneous voltage of the sine input, at time t, is $V = Vpk \sin 2\pi ft$. This may be rearranged to find the highest input signal frequency for an error of ½ LSB (2.44mV) during conversion.

$$v = Vpk \sin 2\pi ft$$

$$\therefore \arcsin\left(\frac{v}{Vpk}\right) = 2\pi ft$$

where v = 2.44mV, Vpk = 10V and t = conversion time

As $\frac{v}{Vpk} \ll 1$, and the sine of a very small number tends to the number itself.

(Remember? $\sin x \doteq x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots\ldots$, so as $x \to o$ the powers of x virtually disappear).

So we are left with

$$\frac{v}{Vpk} = 2\pi ft = 0.244 \times 10^{-3}.$$

Let t = 30us

$$\therefore f = \frac{0.244 \times 10^{-3}}{2\pi \times 30 \times 10^{-6}} = 1.3Hz!$$

A surprisingly low figure, perhaps. By placing a Sample and Hold before the A-D converter, the time t is reduced considerably. Suppose that a S & H samples a signal for long enough to 'acquire' it $\approx$10us for our circuit, and is then switched to Hold, prior to conversion. Just as the device is switched, there is a small window or aperture during which the signal is captured. It is this aperture time which replaces t in the above expressions.

Substituting a typical value of t = 100ns:

$$f = \frac{0.244 \times 10^{-3}}{2\pi \times 100 \times 10^{-9}} = 388Hz$$

This is only food for thought however, because in using Sample and Hold circuits there are many other sources of potential error, especially when a LSB is a small number of mV. These include charge offset, feedthrough, gain accuracy and droop rate. There may also be offset voltages, but these can generally be nulled.

About the analogue components

DG508 – This is a single ended 8 channel analogue multiplexer, which selects one input from eight, according to the 3 address bits. The device exhibits a break-before-make switching action so that 2 inputs cannot be shorted together. It will easily accommodate signal voltages of ±10V with ±12V power supplies. Channel to channel switching time is in the order of 1ys; ON resistance is typically <400R. The DG508 is sourced by several manufacturers.

AD524 – Precision instrumentation amplifier with pin programmable gains of 1, 10, 100 and 1000. This amplifier is included not primarily for its gain but to provide differential inputs. The input and output offset voltages are very small, so that for many applications offset null circuits will not be required. The output of the AD524 will typically settle to 0.01% of a 10V input step in 15ys. The AD524 is made by Analog Devices.

LF398 – Sample and Hold amplifier, although used as a Track and Hold on this board. This is used to capture the input signal when its rate of change is fast compared to the analogue to digital conversion time.

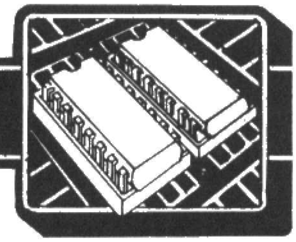
When switched from Hold to Sample, the LF398 should acquire a new signal to within 0.01% in <10ys (with a Hold capacitor of 1,000pf).

LF398 is the National Semiconductors part number. Compatible devices are made by several other manufacturers.

ADC80 – This is a well established, multisourced 12 bit successive approximation A-D converter. It requires very few extra components for basic operation, and without adjustment should be accurate to better than ±0.2% of Full Scale Range (FSR). In this circuit the ADC80 is clocked externally, in synchronism with the CPU clock, and conversion time is 26ys. By changing some links, the ADC80 can be made to perform either 10 or 8 bit conversions in less time.

Note that the converter outputs are inverted. For bipolar operation these are Complementary Offset Binary and for unipolar operation, Complimentary Straight Binary. The analogue input signal range may be selected to be ±2.5V, ±5V, ±10V, 0 to 5V or 0-10V.

The ADC80 should soon be available in a lower cost plastic package.



ZN427 – This 8 bit A-D converter is made by Ferranti and is widely available. It must be clocked externally and requires both gain and offset adjustment. Ratiometric operation is possible with this device.

Conversion time is typically 10ys, but as the processor clock (divided by 2) is used, it will be 18ys in this circuit. Both unipolar and bipolar analogue input ranges are catered for by external resistors.

The outputs of the ZN427 are non-inverting.

DAC80 – An industry standard counterpart to the ADC80, sourced by several manufacturers and already available in a plastic package (low cost), the DAC800P-CBI-V. It is a 12 bit D-A converter with output voltage ranges of ±2.5V, ±5V, ±10V, 0 to 5V and 0-10V. This voltage output can source or sink at least 5mA and will settle to 0.01% of FSR within 5ys for a 20V output swing. (For this performance ±15V power supplies may be necessary).

Next Month – Constructional Details.

Board position 1

F200	Configuration Register (Write)
F201	DAC0 M.S. Byte (Write)
F202	DAC0 L.S. Byte (Write)
F203	DAC1 M.S. Byte (Write)
F204	DAC1 L.S. Byte (Write)
F205	Byte Output (Write)
F206	Spare (Write)
ZF207	Internal Trigger (Write)
F208-F20F	Repeat of above (Write)

Byte input may be read at every even address from F200 to F20E.

F280	Channel 1 M.S. Byte (Read)
F281	Channel 1 L.S. Byte (Read)
F282	Channel 2 M.S. Byte (Read)
F283	Channel 2 L.S. Byte (Read)
F284	Channel 3 M.S. Byte (Read)
F285	Channel 3 L.S. Byte (Read)
F286	Channel 4 M.S. Byte (Read)
F287	Channel 4 L.S. Byte (Read)
F288	Channel 5 M.S. Byte (Read)
F289	Channel 5 L.S. Byte (Read)
F28A	Channel 6 M.S. Byte (Read)
F28B	Channel 6 L.S. Byte (Read)
F28C	Channel 7 M.S. Byte (Read)
F28D	Channel 7 L.S. Byte (Read)
F28E	Channel 8 M.S. Byte (Read)
F28F	Channel 8 L.S. Byte (Read)

Board position 2	F210-F21F and F290-F29F
Board position 3	F220-F22F and F2A0-F2AF
Board position 4	F230-F23F and F280-F2BF
Board position 5	F240-F24F and F2C0-F2CF
Board position 6	F250-F25F and F2D0-F2DF
Board position 7	F260-F26F and F2E0-F2EF
Board position 8	F270-F27F and F2F0-F2FF

Table 1. The analogue board's 1 of 8 switch determines the board number according to the table.

OPTIMAL CODING

James Dick examines various data compressing techniques and examines their application in a variety of systems. We make no apologies for the 'in-depth' treatment of this important subject area.

Communication is one of mankind's most necessary skills and today communication can mean talking to another person or to a machine. The sole aim of communication is to transfer information from source to destination. The language involved may be made up from a descriptive (English) or prescriptive (eg Spanish) dictionary composed of character strings (words) which are, in turn, synthesised from a palette of characters (A . . . Z). The palette is normally representative of the range of available sounds preduceable by the transmitter (speaker) and decipherable by the receiver (listener).

All communications contain a certain amount of redundancy, for example, English, most of the definite and indefinite articles may be omitted without losing meaning. Physiologists imply that most people can still perceive speech when 70% is missing or otherwise corrupted. This is why redundancy is important – in a noisy environment enough of the message is detected to make sense. The Hamming code and other error-correcting codes are examples of controlled redundancy being added to the basic message to improve readability.

Information Theory

Information theory is the name given to describe the results of studying the general communication problem. One section is referred to as source coding, the principle is to use the minimum of characters to convey a message – in a phrase 'optimal coding'. The importance of optimal coding has increased as the electromagnetic spectrum becomes more crowded. As reducing the number of characters in a message increases the capacity of a given channel to transmit data.

Entropy

Entropy is loosely defined as a measure of randomness. The more random a given system is, the higher the system's entropy and this concept may be used to define the efficiency of any system used for optimal coding.

If a given system is expected to transmit K messages where each message is unique and if each of the K messages has a probability of being sent P(i) where i = 1 . . k then if all the messages have equal probability the binary word length is equal to

$$\log_2 k$$

If the probabilities are not equal, the entropy, in bits, is given by

$$E = -\sum_{i=1}^{k} P(i) \log_2 (P(i))$$

In information theory, the entropy is a measure of the information content associated withy a set of messages and gives a lower limit on the number of characters (bits) required to encode the messages. The Average Word Length (AWL) for a set of messages is

NON–UNIQUE CODE

M_1 = 00
M_2 = 01
M_3 = 1
M_4 = 000

Mes. ages 1 to 4 with associated codewords

Wordstring 0001 may be decoded as
M_1 M_2 or as M_4 M_3
and hence is not uniquely decodable

Figure 1. Any coding system must provide 'uniquely decodable' code. This system fails to do this.

SHIFT (ORDER 2) CODE

Message	Message probability	S_2 code	Natural binary
M_1	0.35	00	000
M_2	0.25	01	001
M_3	0.15	10	010
M_4	0.12	1100	011
M_5	0.08	1101	100
M_6	0.03	1110	101
M_7	0.02	111100	110
	Σ = 1.00	AWL - 2.54	AWL = 3.00

Note: The "11" that represents the binary for the 4th message is used as a "label" to indicate a new series of three has started

Figure 2. A shift code in which the last unused basic word is used to indicate an extension.

CONTINUATION BIT CODE (ORDER 2)

Message	Message probability	CB_2 code
M_1	0.35	X00
M_2	0.25	X01
M_3	0.15	X10
M_4	0.12	X11
M_5	0.09	X00X00
M_6	0.04	X00X01
	Σ = 1.00	AWL = 3.39

Inserted continuation bit: may be zero or one depending on value used by last code word sequence

Figure 3. Continuation bit codes are similar to shift codes but are not instantaneous.

TOGGLING OF CONTINUATION BIT

Note: Continuation bit toggles when a new codeword is begun

$\underbrace{\underline{0}\ 0\ 0}_{M_1}\ \underbrace{\underline{1}\ 0\ 0\ \underline{1}\ 0\ 1}_{M_6}\ \underbrace{\underline{0}\ 1\ 1}_{M_4}$

Figure 4. The value of the continuation bit is usually toggled for each word.

FANO CODE

Message	Message probability				Code
M_1	0.35	1	1		11
M_2	0.25		0		10
M_3	0.15	0	1	1	011
M_4	0.12			0	010
M_5	0.09		0	1	001
M_6	0.04			0	000
	Σ = 1.00				AWL = 2.40

Figure 5. Using the Fano code, messages are arranged by probability.

HUFFMAN CODING: PROBABILITY ASSIGNMENT TABLE

Message	Message probability	A	B	C	D
M_1	0.35	0.35	0.35	0.40	0.60
M_2	0.25	0.25	0.25	0.35	0.40
M_3	0.15	0.15	0.25	0.25	
M_4	0.12	0.13	0.15		
M_5	0.09	0.12			
M_6	0.04				
	Σ = 1.00				

Figure 6. Illustrating the rather complex technique of the Huffman code.

$$AWL = \sum_{i=1}^{k} WL(i)P(i)$$

where WL(i) is the word length in characters for the i-th message. Optimal coding attempts to get AWL close to E. The efficiency of a coding technique is often given as

$$\text{Efficiency} = 100 \times \frac{E}{AWL} \%$$

Codes

The previous sections have shown that a transfer of information involves the transmission of messages each with a certain probability of being used and each with a certain number of characters. It is intuitively obvious that high probability messages should be assigned a smaller number of characters than low probability messages. Adopting this technique produces 'unequal length' code.

Any coding system must provide 'uniquely decodable' code. This means that any series of words may only be decodable in one way. **Fig 1** shows a code which fails to do this.

Shift Codes

Define a shift code of order n to use words n bits in length as the basic high probability word-length. The choice of n will depend on the number of messages with a high probability. The basic word allows 2^n code words. The first (2^n-1) messages are given codes, usually in natural binary sequence. The last and unused basic word is used to indicate an extension. **Fig 2** shows an example.

Continuation Bit Codes

These are particularly suitable where message probability obeys a power law, e.g.

$P(m_i) = i^{-c}$

where

$P(m_i)$ is the probability of the ith message and
c is a positive constant

Each message is given a $(n+1)$ length code where n is the order of the continuation bit code used. The extra bit, usually inserted at the most significant end of the code word, indicates whether or not the next word is an extension. **Fig 3** shows a B_2 code; the similarity with the shift system is very evident. The shift codes are instantaneous while the continuation bit codes are not: the next bit has to be examined to determine whether the end of a particular word has been reached. The value of the continuation bit is normally toggled for each word. See **Fig 4.**

This type of code is easily implemented in hardware.

Fano And Huffman Codes

The Fano code, and the Huffman system described below, are two nearly-optimal codes with efficiencies commonly greater than 75%. At first sight they appear very similar but subtleties in the latter give it greater efficiency.

For the Fano code, the messages are arranged by probability as shown in **Fig 5.** The groupings in (0, 1) pairs gradually build up the code words.

The Huffman code is similarly constructed. A table of messages and their probabilities is made. To proceed from left to right, the two smallest probabilities are combined and re-entered in the next column until there are only two probabilities left. The code words associated with each message are then generated by traversing the table in the retrograde direction assigning a bit value of 0 to the probability which was the result of an addition in the next (ie leftmost) columnm. **Fig. 6** illustrates this rather complex coding technique.

Next month we look at some practical applications of the ideas expressed here including a look at Image Storing using such techniques as Run Length Coding and Differential Pulse Code Modulation.

IN PRAISE OF ZX BASIC

Mike James recently realised that, while he liked ZX BASIC, he rarely owned up to the fact in 'respectable' company. Here he sets the record straight by pointing out how logical and straightforward the Sinclair dialect of BASIC is.

When the now almost forgotten ZX80 first appreared one of its least pleasing features was its very limited integer-only BASIC that barely provided enough computing power to do anything useful, no matter what the manual might say. I think that this poor start for what has now become a powerful and very logical dialect of BASIC is responsible for the lack of attention, and indeed even derision, that ZX BASIC has received. There is perhaps a natural tendency to ignore the features of a programming language when it is available on only one or two very similar machines. In this sense comment and discussion of ZX BASIC are usually confused with a wider debate about the merits and demerits of the ZX81 or the Spectrum. Now while it isn't possible to ignore completely the hardware that a language runs on, it is possible to identify areas where the language helps or hinders the easy use of the machine.

A Little History

Before going on to explain the features of ZX BASIC that make it such a good incarnation of the language it is necessary to explain a little of the effect that the major dialect of BASIC – Microsoft BASIC – has had. In the early days of microcomputing there was no clear cut definition of BASIC and many manufacturers spent time producing their own limited implementations that served to sell machines but were not really suitable for writing programs. When Microsoft BASIC became available on a range of microprocessors, and in a ROM form on machines such as the APPLE, the PET and the TRS-80 Model I, it was received with enthusiasm. It was powerful enough to allow useful programs to be written while being compact enough not to occupy too much memory and it set the standard for a minimum BASIC.

It is very easy to underestimate the success of Microsoft BASIC by assuming that BASICs like APPLESOFT, PET BASIC, TRS-80 BASIC, Atari BASIC, Dragon and Tandy Color BASIC are all different when in fact they are all customised versions of Microsoft BASIC in various stages of development. Dialects of BASIC that came after Microsoft BASIC usually carry a note to the effect that they are Microsoft-compatible or at the very least Microsoft-like.. This has restricted new dialects of BASIC to extensions of Microsoft BASIC. In some ways this is an advantage in that it means that programs are more portable but it also means that any early design mistakes in the language are fixed forever.

ZX BASIC is the one dialect that has made some significant changes, in particular to the way that strings and string arrays are handled, without also providing the equivalent Microsoft commands. In my opinion the way that ZX BASIC handles strings is one of its best features and its method should become the standard. Although it is ZX BASICs string handling methods that are of most interest to users of other dialects of BASIC, the first topic to consider is the central principle behind any BASIC – expression evaluation.

The Expression Evaluator

The first high level language, FORTRAN, introduced the idea of an 'arithmetic expression' – a valid formula involving the usual operations of arithmetic and a range of functions – and of expression evaluation – methods of reducing an expression down to a single value. Expressions and expression evaluation have now become so familiar to any high level language programmer that it is all too easy to take them for granted. Expressions and evaluation are, however almost the only opportunities that high level languages give programmers for changing data. Without expression evaluation, computing would be limited to moving data from one place to another. In some senses the power and ease of use of a language come from the range of data and operations that can be combined together to form expressions and the absence of restrictions on where you can use such expressions. An ideal dialect of BASIC would allow all types of data and all types of operation to be combined whenever it made sense. In fact it is not quite as simple as this in that this requirement can easily be met by claiming that all but the most obvious operations are nonsensical. For example, what does –

```
3+(2>0)
```

mean? It would be all too easy for the author of a dialect of BASIC to save time and trouble by ruling that such an expression is nonsense. In ZX BASIC (and in later versions of Microsoft) though such expressions are quite valid because the result of any relation is a number which represents one of the two possible results – ie true or false. In ZX BASIC true is represented by 1 and false is represented by 0 so evaluating the above expression gives 4.

Nearly all versions of BASIC recognise the types of expression shown below.

```
1. arithmetic expressions e.g. A+3*SIN(X
2. relational expressions e.g. A<>3
3. Boolean or logical expressions
   e.g. A AND B
4. string expressions e.g. A$+"123"
```

The way that BASICs differ is in what data can be involved in each type of expression and how the types of expression can be mixed. For example, some versions of BASIC only allow Boolean expressions to involve the results of relational expressions. That is (A<B) AND (B=0) is valid but A AND B is not. ZX BASIC selects its data values and data types to make it possible to mix expressions as freely as possible. In particular, arithmetic expressions do not distinguish between integer and real values although a special representation is automatically used for integers for reasons of efficiency. Relational expressions can involve both arithmetic expressions and string expressions although it obviously makes no sense to compare a string to a number! The result of evaluating a relational expression is an integer – 0 for false and 1 for true. Boolean expressions can involve any other expression that evaluates to a number. In this case 0 represents false and any other positive value (including 1) represents true. ZX BASIC is distinct in the way that it implements its Boolean operations on values other than 0 and 1 in that it treats the right and left hand values differently. This is shown below.

```
        x AND y   evaluates to   x if y is true (i.e. y<>)
                                 0 if y is false (i.e. y=0)
and
        x OR y    evaluates to   1 if y is true (i.e. y<>0)
                                 x if y is false (i.e. y=0)
```

Notice the way that the value of x effects the result in a different way to the value of y. If you are familiar with the more usual 'bitwise' logical operations of other versions of BASIC this strange way of interpreting AND and OR may seem of little use. Other versions of BASIC make their logical operators available for 'bit manipulation' by applying the operation to each pair of bits in the binary representation of values involved. That is for most versions of BASIC

```
4 OR 2
```

evaluates to 6 because 4 is 100 in binary and 2 is 010 and ORing these together gives 110 or 6. However, in ZX BASIC –

```
4 OR 2
```

is 1. This inability to do bit manipulation is very important if you are interested in controlling hardware etc and at first sight the strange way that ZX BASIC handled Boolean operations on values other than 0 and 1 was a failing. Most BASIC programmers are however not concerned with bit manipulation (readers of *E&CM* are possible the exception in this!) and on closer examination ZX BASIC does provide something else of use to them. The result of evaluating AND could be written –

```
x AND y = IF y THEN x ELSE 0
```

and similarly

```
x OR y = IF y THEN 1 ELSE x
```

Notice that both of these definitions do give the correct answer for AND and OR if x and y are restricted to 0 and 1. In this sense they are just as valid as the more familiar bitwise operations. When written in the IF . . THEN . . ELSE form it should be easy to see what service they can provide the BASIC programmer with. For example, in ZX BASIC you can write expressions such as –

```
COST = TIME*(RATE AND R$=
"STD") + TIME*(HIRATE AND R$=
"PEAK")
```

If R$="STD" then the first bracket works out to RATE and the second bracket works out to 0 making the whole expression evaluate to TIME*RATE. However, if R$="PEAK" then the first bracket is 0 and the second bracket is HIRATE making the whole expression TIME*HIRATE. In other words if you have two mutually exclusive calculations then you can combine them into one 'conditional arithmetic expression'. OR can be used in a similar way. It is possible to write conditional arithmetic expressions using other versions of BASIC but ZX BASIC seems to encourage them, so extending the idea of an expression to include conditionals.

It is equally surprising to discover that for one Boolean operator, AND, the range of data and results extend to strings. In ZX BASIC X$ AND Y will evaluate to X$ if Y is true (ie any non-zero value) and to the null string if Y is false (ie zero). You can see that this is a logical extension of the way that AND normally works, with string values playing the role of true and the null string playing the role of false. String expressions can also be mixed with arithmetic expressions by use of the VAL function, which changes a string of digits into a number. Finally arithmetic expressions can be mixed with string expressions by the use of the STR$ function which converts a number to a string of digits. (Functions like VAL and STR$ that convert one data type to another are usually called transfer functions).

All in all ZX BASIC goes out of its way to try to allow as much calculation and data manipulation to be combined in a single expression as possible.

Using Expressions

ZX BASIC first begins to show its lack of restriction in the use of expressions. An ideal BASIC would follow the principle that anywhere that you can use a constant you should be allowed to use an expression of the same type. After all what is a constant other than the simplest form of an expression? However, you would be surprised how many versions of BASIC break this rule in some way or other. In most cases these restrictions are simply the result of sloppy interpreter design, in others there has been a genuine attempt to prohibit bad practice. In ZX BASIC the rule is applied almost without exception. For example, it is well known to ZX BASIC programmers that the general form of the GOTO and GOSUB instructions are –

```
GOTO expression
GOSUB expression
```

so statements like –

```
GOTO INT(RND*10)
```

are quite reasonable in ZX BASIC! It is usually claimed that the main reason that ZX BASIC allows expressions in GOTOs and GOSUBs is to make up for its lack of ON . . GOTO and ON . . GOSUB instructions. This is certainly true but it also allows you to write instructions like

```
GOSUB PRINTOUT
```

where PRINTOUT is a variable that contains the line number of a subroutine that prints something out. Apart from problems it creates for renumbering, this technique does improve the readability of programs and could be used more often.

A second and slightly less well known example of how ZX BASIC allows expressions to be used freely is provided by the INPUT statement. To input a value ZX BASIC uses the whole of its powers of expression evaluation. For example, if yuou try the following simple program –

```
10 INPUT A
20 PRINT A
30 GOTO 10
```

and enter 2*2 in response to the INPUT you might be surprised to find that this expression is evaluated and 4 is printed by line 20. This can be a very useful feature. For example, when using a technical program, you could enter data with a multiplier to convert the data to the correct units. However, just think of the trouble that it could cause in a program that is trying to teach arithmetic – the user could short circuit the learning process and score 100% by typing the questions back in! An even more potentially dangerous problem, if you are not aware of it, is illustrated by adding –

```
5 B=10
```

to the program above. Now if you enter something like B*2 in response to the INPUT prompt you will discover that expression evaluation extends to looking up the current value of B! This is extremely useful when testing a program but can lead to disastrous consequences if the user keys in a variable name by accident. The same full expression evaluation is applied to the INPUT of string data. Once you know about this facility it is not difficult to find ways of actually making use of it.

The only two places that I know of where ZX BASIC treats constants and expressions on a different footing are BASIC line numbers and the BIN function. (The BIN function is only found in the Spectrum version of ZX BASIC). It is possible to imagine a system where line numbers could be specified by expressions but I think that there are easier ways of going about things! The function BIN, however, falls into a different category because it would be useful to be able to use BIN to convert binary expressions to decimal. However, ZX BASIC expressions do not include binary numbers as a data type and so the BIN function is treated all on its own and can only be used with constants.

As well as using its expression evaluator every time a value is required, ZX BASIC also allows the programmer to use it. The function VAL(string expression) will evaluate any arithmetic expression contained in 'string expression'. So for example, you can tabulate a user supplied function using –

```
10 INPUT "Enter a function of X ";A$
20 FOR X=0 TO 1 STEP ,05
30 PRINT X,VAL(A$)
40 NEXT X
```

This will print the value of any function for values of X in the range 0 to 1. You could enter SIN(X) or X**2+X+3 and the VAL function would evaluate the expression each time through the loop using the current value of X.

Similarly, on the Spectrum the function VAL$(string expression) will evaluate the string expression contained in 'string expression'! The VAL$ function is just as easy to use as the VAL function but it tends to get a little confusing because of the way that the term string expression is used twice. For example

```
VAL("2+2") will evaluate 2+2 (i.e. 4)
VAL$("""A""+""B""") will evaluate
    to "AB"
```

The VAL$ example looks very complicated because in ZX BASIC a quote must be written twice if it is inside another pair of quotes. If you remove the doubling quotes then the expression to be evaluated is simply "A"+"B". The VAL$ function really only comes into its own when used to evaluate a string expression stored inside a string variable. For example,

```
A$="X+2"
VAL(A$) will evaluate X+2
```

and

```
A$="B$+""S"""
VAL$(A$)
```

will evaluate B$+"S". In other words, it will concatenate the current contents of B$ with

"S". The VAL$ function is nowhere near as useful as the VAL function. However, it sometimes can come to the rescue in the most unlikely situations. For example, while writing a memory dump program I found that it was easy to find the names of all the string variables that a program had used but not their contents. However, once the name of a string is stored in A$ its value can be obtained by VAL$(A$)!

ZX BASIC and Strings

Microsoft BASIC has set the standard for string handling by way of the functions LEFT$, RIGHT$ and MID$. However, the whole idea of using functions to extract substrings is quite unnecessary. In ZX BASIC a string is treated as if it was a variable length array of characters – which is exactly what a string is. For example

if

```
A$="ABCDEF"
then
A$(1) is "A"
and  A$(4) is "D"
```

This notation is familiar from standard numeric arrays. ZX BASIC also allows 'proper' arrays of characters to be defined – For example

```
10 DIM B$(10)
```

defines a one dimensional array B$ consisting of exactly ten characters. Notice that the only difference between A$ and B$ is that B$ is fixed in length. Assigning a single character to an array of characters or to a string is logically the same as assigning values to a numberic array. For example, both

```
A$(2)="A"
and
B$(5)="X"
```

assign characters to single elements of a string and a character array respectively. However, it is more convenient to assign a group of characters at a time and so the usual single element array assignment is extended to give the standard string assignment. So,

```
A$="ABC"
```

is the same as

```
A$(1)="A":A$(2)="B":A$(3)="c"
```

This extended form of assignment is where the difference between a string and a character array really shows itself. Firstly a string will accept any number of characters (up to a maximum of 255) and a character array will only accept the number of characters that it was dimensioned to accept. For example

```
A$="ABC"
```

produces a string of three letters but

```
10 DIM B$(10)
20 B$="ABC"
```

sets the first three characters of B$ to ABC and the last seven to blanks. To assign values to a part of a string or a character array needs only another a small extension to the usual array index notation. For example,

```
A$(3 TO 5)="ABCDEF"
```

will assign A to A$(3), B to A$(4) and C to A$(5). This notation is called a 'slicer' and it can also be used to specify part of a string in other situations. For example,

```
10 A$="SPECTRUM"
20 PRINT A$ (3 TO 5)
```

will print "ECT". Apart from a few extra notational conveniences, such as

```
A$ (TO 3) meaning A$(1 TO 3)
and
A$(3 TO) meaning A$(3 TO LEN(A$))
```

this is all there is to ZX BASIC string and character array handling. In many ways the use of slicers is a more direct way of specifying substrings, string elements etc. than the equivalent Microsoft functions MID$, LEFT$ and RIGHT$.

Moving on to two and more dimensional character arrays is just as easy using the slicer notation although a slicer can only be used on the last index. So for example,

```
DIM A$(10,10)
```

defines a 10 by 10 character array and A$(3,3) is the third character in the third 'row'. Using the slicing notation you can specify substrings such as –

```
A$93,4 TO 6)
```

which corresponds to character 4 to 6 in the third 'row'. Leaving the final slicer out is the same thing as specifying its maximum limits. That is A$(3) is the same thing as A$(3,1 TO 10).

ZX BASIC string handling brings out the relationship between strings, character arrays and numeric arrays. The slicing notation is a more direct way of specifying substrings and it makes changing sections of existing strings particularly easy. For example, compare the clumsy and time-consuming

```
A$=LEFT$(A$,2)+"XYZ"+
RIGHT$(A$,LEN(A$)-5)
```

which is the Microsoft way of replacing the 3rd, 4th and 5th characters in A$ with "XYZ" with the ZX BASIC –

```
A$(3 TO 5)="XYZ"
```

Friendly And Versatile

In this short look at ZX BASIC the emphasis has naturally been on the few advanced or novel features that can be described in isolation. In use it is the way that these features fit together to give a friendly and economical language that makes ZX BASIC suitable for beginner and expert alike. There is also much to be said for the logic behind the Spectrum's parallel attribute graphics system and the way that ZX BASIC makes this available to the user. However, this is the sort of area where it is difficult to see how much praise should fall on the hardware designer for recognising iust what is needed to produce colour graphics displays and how much to the software designer for making good use of what was provided. It's rather like trying to comment separately on the words and melody of a song and failing to notice that they fit together! There are plenty of features that I would like to see added to ZX BASIC, including an ON ERROR GOTO statement, a procedure call and local variables, but these would make ZX BASIC almost too big to fit into its current 16K ROM. Once again software and hardware interact! Seen as a whole ZX BASIC isn't perfect but it does deserve more praise than it currently gets.

E&CM

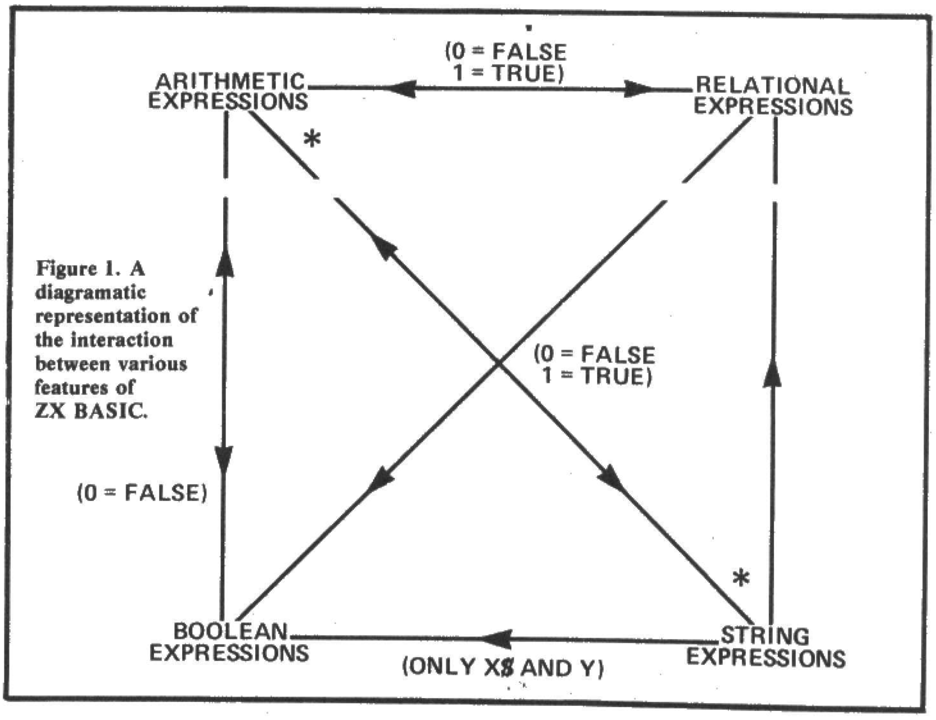


Figure 1. A diagramatic representation of the interaction between various features of ZX BASIC.

MICRO GRAPHICS TECHNIQUES

Part 1 NEW SERIES

Mike James begins a new series that will explore the world of computer animation. Whether your interest lies in CAD or in games the principles developed during the series should enhance your program's output. This month – Sprites, the elements of animation.

The fact that micro computers have brought to graphics the status of a popular pastime is obvious. Historically graphics have either been taken very seriously, as in computer aided design, or have been treated as a source of fun, as in computer games. For the future it is becoming clear that graphics are going to play a much more central role in the use of computers. The new operating systems and applications software on machines like Apple's new Lisa interact with the user almost exclusively through graphics. Some of the problems of the high cost of good quality graphics hardware are going to be solved by the new 16-bit and 32 bit micros just coming off the production lines and drawing boards. However, even though graphics are so important, and despite the fact that a lot of programmer time and attention has been devoted to the subject, it is still not easy to see how the whole area is anything more than a collection of special techniques.

In this series of articles the objective will be to present the ideas and techniques underlying graphics on a small machine. The trouble with small machines is that they lack the computing power necessary to carry out the complicated calculations often involved in grapics in a reasonable amount of time. Very often even moving the requisite amount of data from one place to another can be too much for a micro! These difficulties have to be seen as something of a contradiction when even the lowest cost micro has high resolution colour graphics that not so long ago would have cost a small fortune. The result is that even if you resort to assembly language programming it is still rather more difficult to produce effective graphics than manufacturers' literature and hardware specifications might lead you to believe. Currently micro graphics is a mixture of theory and special 'go faster' tricks based on everything from using the hardware in ways that was never intended to fooling the viewer using a little simple psychology!

It would be possible to start a series on graphics by explaining the mathematical theory that lies behind 2D or 3D representations of objects or screens but it would take some time before anything practical or rewarding was achieved. So to kick off with something more immediately useful, this first part of the series looks at the theory and practice, of animation or 'Micky Mouse' graphics as it is sometimes called. Although the ideas and methods described throughout the series can be used on any micro with reasonable graphics facilities it is impossible to avoid being machine specific in some cases. The machines that will be specifically kept in mind during explanations and examples are the Spectrum, the BBC Micro/Electron and the Dragon/Tandy Color.

Small Shapes

It is worth making the distinction between large scale shapes drawn using lines, circles and colour filling instructions from small scale shapes. Small scale shapes are produced using a small number – generally less than a hundred – of display points or 'pixels' (short for 'picture elements'). The fact that so few points are involved means that it is possible simply to 'quote' the shape at any location where it is needed. The number of points involved in a large shape usually makes it worth replotting it using commands that draw lines, circles etc. every time it is needed.

Small shapes are so important that most computers provide special ways of quoting them. In most cases this is just an extension of the method used to print text characters on the screen to include user-definable graphics characters (eg Spectrum and BBC Micro). However, it is important to realise that user-definable characters form just one way to produce small graphics shapes. For example, the Dragon and the Tandy Color use two extra commands, GET and PUT, to define and reproduce small shapes. The GET command will store the dot pattern of any part of the screen in an array and the PUT command will transfer it back to any location.

The disadvantage of thinking about small shapes as the same as text characters is that there is a tendency to use them with the same set of restrictions. For example, it is common to print text only at a fixed number of locations on the screen, the so called 'character locations', but it is better to be able to print small shapes anywhere on the screen. The Spectrum, for example, will allow you to define any shape that you like within an 8 by 8 square of pixels but you can only print this shape into one of the 32 columns by 22 lines normally used for text characters. (This is in fact a software limitation rather than anything to do with the hardware!). The BBC Micro, however, allows the same 8 by 8 character block to be printed anywhere in the high resolution screen.

Whatever method a computer uses to produce small shapes, they have two main uses. The first is to fill the screen with multiple copies of a shape – for example, you could define the shape of a house and produce a small town by printing it in different places and colours, and the second is in animation. For both applications the important factor is speed. It is very important that whatever computer you are using for graphics it must be able to produce small shapes very fast. for many applications this is a more important feature than good high resolution graphics commands.

Before leaving the subject of small shapes it is worth mentioning the way that the remarkable Atari range of personal computers deal with small shapes. In most computers the definition of a user-defined character is held in memory and every time you print the character the machine transfers the pattern of pixels to the screen RAM, which is of course a different area of memory. In this way printing (or in the Dragon's case PUTting) a small shape is simply a matter of transferring the bits that represent the dot pattern from one area of memory to another. This means that once a character is printed you can change its definition without changing anything that is already on the screen.

The Atari however takes a completely different approach. The definition of a user-defined character is held in memory in the usual way but instead of transferring the pixel pattern to another area of memory when it is printed the Atari simply transfers a character code which indicates which character is to be displayed. This code is used to look up a character definition each time that the screen is displayed. So if you change the definition of the character then all the existing examples of it already on the screen instantly change their shape! This can obviously be used to produce internal movement in a small shape, but more of this later.

Blanking Animation

The standard method of making things move in a computer display is fairly well known even to novice programmers. To move a shape on the screen you first draw it at one location, then remove it or 'blank' it out, only to redraw it slightly shifted. The smoothness of the apparent motion produced by this blanking animation depends on a number of interacting factors. The following guidelines should aid the production of smooth movement.

The distance moved at each step should be small.

the time taken to blank and produce the shape should also be small.

The shape should be displayed on the screen for as long as possible between moves.

For really smooth motion each step should be synchronised with the TV frame display rate.

The perceived smoothness is also affected by the shape, colour and texture of the object.

The trouble is that in practice most micros are too slow to produce smooth animation using BASIC. Even using assembler, a great deal of care and attention is needed to produce a flicker-free screen. The problem increases as the number of separate objects being moved increases. Normally animation graphics are developed in a haphazard way, with each moving object being added to the program in turn perhaps even using a completely different method for each. However, there is a way of organising the animation of any number of small shapes that is entirely systematic.

Sprites

Once you start writing animation programs you soon learn that the best way to think of the path that a moving object takes is by using the idea of velocities. For example, if an object is moving smoothly in a straight line across the screen it can be associated with four quantities –

X – its present x co-ordinate
Y – its present y co-ordinate
XV – its x or horizontal velocity
YV – its y or vertical velocity

In other words at any given point in the program the object is at the position X,Y and its next position will be X+XV,Y+YV. The use of velocities makes it particularly easy to update the current position to give the new position using nothing but addition – which should be one of the fastest operations that a micro can perform. In practice moving an object about the screen rarely involves nothing but motion in a straight line! Fortunately it is not difficult to extend the use of velocities to produce all types of movement.

A small graphics shape together with its current position and a pair of velocities is usually called a 'sprite'. (They are also known as 'MOBs' standing for 'Movable OBjects'). This sort of sprite is in fact the simplest used in animated graphics and in practice the idea is elaborated considerably. Sprites are often associated with special graphics hardware that will automatically update positions and plot the shape on the screen. However, while sprite hardware does make things easier and faster, software sprites are also useful in writing programs on any machine.

Acceleration

Before giving an example of how sprites can be used it is necessary to add the idea of acceleration. As already noted it is rare that objects move smoothly in straight lines. For one thing they tend to go off the edge of the screen! In practice objects move across the screen changing their velocity, sometimes continuously as in the case of a ball or a lunar lander 'falling' in an arc down the screen, and sometimes suddenly as in the case of an object 'bouncing' off the sides of the screen. However, the common factor is that the velocity does change and a change in velocity is usually called an 'acceleration'. This suggests that to make sprites really useful the number of quantities associated with a sprite should be increased to six by adding a horizontal and a vertical acceleration. —XA and YA. Now at each stage the current position is updated by

X = X + XV
and
Y = Y + YV
and the velocities by
XV = XV + XA
and
YV = YV + YA

This method does indeed work for movements that involve smooth changes in an object's velocity but what about the sudden changes involved in an object 'bouncing' off another object? To take this sort of sudden change into account we have to introduce yet another idea– that of a 'force function'. A force function defines the way that the accelerations depend on the current position, the current velocity, the current acceleration and any other conditions that you might want to take into account! It is easier to understand the way that a force function controls the movement of a sprite by looking at an example. But first it is worth summarising the definition of a simple sprite. A sprite is composed of the following information –

1 – The shape of the object
2 – The current position stored in X and Y
3 – The accelerations XA and YA
4 – The force function that is used to update accelerations

Sprite Example

Consider the well known problem of bouncing a ball around the screen in terms of sprites – except that to show the advantages of sprites consider bouncing more than one ball around the screen. In fact the program given below will bounce any number of balls around the screen if you don't worry about how fast (or rather slow) everyting moves!

The first part of constructing the program involves defining a suitable sprite for the bouncing ball. The shape of the ball is easily solved and we can assume that it is stored in the string S$. In other words, PRINT S$ produces the ball shape on the screen. Its current position can be stored in X and Y and its current velocity in XV and YV. The problems really only start when you try to work out what the acceleration should be. When the ball is moving about the screen away from the wall the answer is easy – both horizontal and vertical acceleration are 0. However, when the ball is near, or to be more accurate touching, the edge of the screen the acceleration is clearly not zero because the one of the ball's velocities will be reversed to create the bounce. If the ball meets a horizontal wall then YV is changed to —YV, if it meets vertical wall then XV is changed to —XV. In terms of acceleration this implies –

```
IF the ball is in contact with a horizontal wall THEN
  YA=—2*YV ELSE YA=0
IF the ball is in contact with a vertical wall THEN
  XA=—2*XV ELSE XA=0
```

To turn these two IF statements completely into BASIC only requires the test for horizontal and vertical walls to be made exact. If we consider that the screen co-ordinates go from 0 to 31 horizontally and 0 to 15 vertically then the tests can be written –

```
IF Y=0 OR Y=15 THEN YA=—2*YV ELSE YA=0
IF X=0 OR Y=31 THEN XA=—2*XV ELSE XA=0
```

You should recognise these two IF statements as the force function for the ball because together they determine the acceleration at each update.

Now all that remains is to put all of the parts of the sprite together to produce the program. However, the whole point about using sprites is that once you have defined a sprite for an object it is easy to re-use the definition to produce any number of examples of the sprite moving around the screen. All that you have to do is to replace each of the

variables associated with the sprite by an array large enough to hold the information for each sprite. For example, if you want five balls bouncing round the screen the use X(5) to record the current x co-ordinate of each sprite – X(1) is the x co-ordinate of the first example of the sprite, X(2) the x co-ordinate of the second and so on. The resulting program for the BBC micro or Electron is shown in **Program 1.**

```
10 GOSUB 1000
20 PRINT "How many sprites";
30 INPUT N
40 DIM S$(N),X(N),Y(N),XV(N),YV(N)
  ,XA(N),YA(N)
50 DIM PX(N),PY(N)
60 GOSUB 1000
70 GOSUB 2000
80 GOSUB 3000
90 GOTO 80

1000 MODE 6
1010 RETURN

2000 GOSUB 6000
2010 FOR I=1 TO N
2020 S$(I)=CHR$(240)
2030 X(I)=INT(RND(0)*30)+1
2040 Y(I)=INT(RND(1)*14)+1
2050 PX(I)=X(I)
2060 PY(I)=Y(I)
2070 XV(I)=SGN(RND(1)-.5)
2080 YV(I)=SGN(RND(1)-.5)
2090 XA(I)=0
2100 YA(I)=0
2110 NEXT I
2120 RETURN

3000 FOR I=1 TO N
3010 PX(I)=X(I)
3020 PY(I)=Y(I)
3030 X(I)=X(I)+XV(I)
3040 Y(I)=Y(I)+YV(I)
3050 IF X(I)=0 OR X(I)=31 THEN XA(I)=-2*XV(I) ELSE XA(I)=0
3060 IF Y(I)=0 OR Y(I)=15 THEN YA(I)=-2*YV(I) ELSE YA(I)=0
3070 XV(I)=XV(I)+XA(I)
3080 YV(I)=YV(I)+YA(I)
3090 PRINT TAB(PX(I),PY(I));" ";
3100 PRINT TAB(X(I),Y(I));S$(I);
3120 NEXT I
3130 RETURN

6000 VDU 23,240,&FF,&FF,&FF,&FF,&FF,&FF,&FF,&FF
6010 RETURN
```

PROGRAM 1

Subroutine 1000 clears the screen. Subroutine 2000 sets initial values for each of the sprites variables. Random values are used for the initial positions and velocities. Subroutine 3000 carries out the updating of position, velocity and acceleration. Subroutine 6000 sets the shape of the small solid object.

On the BBC Micro the program can animate around five balls without too much loss of speed and the resulting display is quite impressive. Users of the Spectrum and the Dragon should have no difficulty in converting the program because none of the extra special features of the BBC Micro have been used but the number of balls that can be bounced is likely to be fewer than five. Spectrum users should change the PRINT TAB to PRINT AT and Dragon users should use PRINT @ in the usual way.

The power of the sprite method of organising animation programs should be clear from the economy and simplicity of the above program. However, the sprite method also leads to programs that are easy to modify. For example, if you want to add some 'gravity' to the bouncing balls so that they fall in a parabolic arc and then bounce back up to the same height change line 3060 to –

```
3060 IF INT (Y(I))<=0 OR INT(Y(I))>=15 THEN
YA(I)=-2*YV ELSE YA(I)=.1
```

Apart from changing the tests for a collision with the boundary to take account of the fact that Y(I) may be fractional, the main change is to set the vertical acceleration to a small constant to mimic the effect of gravity.

Other changes are equally simple. For example, if you want to make the balls bounce less each time change line 3060 to –

```
3060 IF INT (Y(I))<=0 OR INT(Y(I))>15 THEN YA(I)=
-1.99*YV ELSE YA(I)=.1
```

where the 1.99*YV reflects a reduction in the vertical velocity on each (horizontal) bounce). You could make each of the sprites have a different shape by changing subroutine 2000 where S$ is defined. You could even attach a colour code, a sound to be produced at each bounce etc. to the sprites.

Priorities, Collisions And Events

You may be thinking that bouncing balls around the screen is hardly a job of work by which to test the usefulness of the sprite idea. However, once you start exploring some of the additional ideas that suggest themselves and seem natural when you work with sprites it becomes difficult to believe that you ever thought about animation in any other way! For example, the order in which you blank and draw sprites imposes a natural priority on a collection of sprites. A later sprite will appear to pass in front of an earlier sprite if they both happen to reach the same screen position. This is simply because the later sprite will overwrite the earlier one but it leads to a way of ordering sprites so that when they meet you can predict what the result will look like. The idea of one sprite passing over another one leads naturally on to the idea of sprite-sprite collisions. At the end of all sprite moves it is easy (but time consuming in BASIC) to check to see if any of the sprites are in the same position and so have collided. A sprite collision is usually dealt with by a special subroutine. for example, if one of the sprites is a rocket and the other a target then when a sprite-sprite collision is detected the obvious thing to do is to call a subroutine that produces an explosion! This idea of a collision causing something different to happen within the program is very like what happens during a hardware interrupt. Interrupts are normally used to inform the computer that something unusual or important that needs special attention has happened in the outside world. This is exactly what happens with a sprite collision except that the event is internal.

The idea can be generalised to sprite events which correspond to any detectable condition that should cause the program to do something different. For example, the sprite bounce from the boundary wall that was controlled by the force function could have been declared a sprite event that when detected called a subroutine that reversed the velocity and perhaps made a noise. There is often more than one way to animate a sprite and the number of ways increases as the idea of a sprite becomes more and more elaborate. Some of these ideas will be treated later in the series but there is a lot to be gained from trying to keep sprites simple – especially in BASIC!

Without Going Anywhere

You may already be trying to think of examples of moving graphics that you have seen that would be difficult to implement using sprites. The majority of animated displays are not difficult to interpret in terms of nothing but sprite graphics once you have had a little practice. For example, consider the traditional space invaders screen with alien ships in rows moving from side to side and then slowly moving down the screen. How can this almost static movement be translated into sprites? The answer is surprisingly easy! Change the following lines in the original bouncing balls program –

```
2030 X(I)=I*3
2040 Y(I)=1
2050 XV(I)=2
2060 YV(I)=.1
3050 XA(I)=-2*XV(I)
```

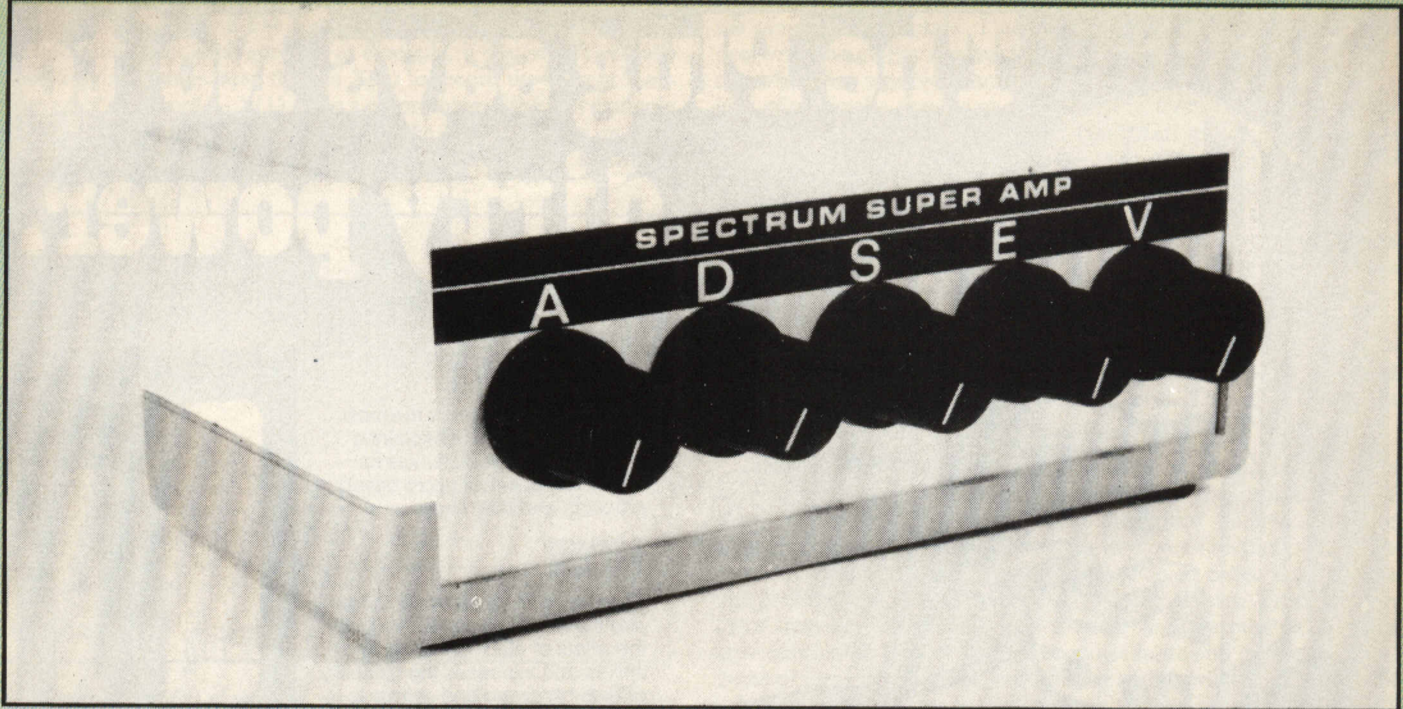
The only changes are to the initial positions and velocities of the shapes and to the force formula. The initial positions are set to produce a single row of shapes by line 2030 and 2040. Line 2050 sets a horizontal velocity of 2 and a vertical velocity of .1. As you might expect, this results in a large sideways velocity but a very small vertical velocity making the row creep forward. Line 3050 is responsible for the side to side oscillation because it reverses the velocity at each update. You should now be able to see how this method could be extended to more than one row and how to make the 'ships' move faster down the screen as the game progresses without having to use a single non-sprite idea!

Conclusion

The main topic of this first part of Micro Graphics has been the sprite. This powerful animation method provides a good theoretical background to use when constructing programs. However, from BASIC a program that is constructed using sprites will often run too slowly to produce a good effect. In this situation there is no choice but to resort to 'dirty tricks'. The bouncing balls program can be made to go faster by tackling the problem directly but even then there is something to be gained from using sprites in the design stage. However, if you are using assembly language then sprites are an ideal method of organising and implementing your program.

Next Month – Sprites make a second appearance when small moving shapes also change their shape as they move, crash into other shapes and generally explode!

E&CM



SPECTRUM EFFECTS UNIT

This project uses some cunning circuitry to vastly enhance the Spectrum's BEEP sound command. It also incorporates an amplifier capable of filling a room with the resultant effects. Mark Stuart describes the design.

The Spectrum Sound Board project published in our August and September issues showed one of the ways in which the Spectrum's sound generating capability could be enhanced. The effects unit described here shows an alternative approach that has the advantage of being both easy to use and capable of producing a wide range of interesting effects far removed from the underlying 'beep'.

Five controls are provided on the unit, these being a volume control, and four effects controls which modify the envelope of the sound. Three of the effects controls set the 'attack', 'sustain', and 'decay' characteristics of the sound. A fast attach and slow decay setting gives a percussive sound while an envelope with a slow attack and short decay sounds more like a wind instrument. The Sustain control determines the duration of the sound, from the start of the attack period to the start of the decay period. The three envelope controls allow the sound to be modified over a very wide range, producing excellent sound effects for games, or musical applications.

The fourth effect control governs the echo effect that is produced by interrupting the output at a rate set by the control. The echo effect can be turned off using a switch on the echo control.

In order to keep the amplifier as easy as possible to use, it is powered by an internal PP3 battery. The only connection required is from the Spectrum 'mic' socket to the amplifier input, using the tape input/output lead. For further simplicity the amplifier on/off switch is incorporated into its input jack socket. The power is switched on by inserting the input jack plug.

To enable a PP3 battery to have a good working life great care was taken to minimise standing current. In the absence of an input signal all the amplifier stages are biased off. A single CMOS IC is used to provide the switching functions.

Envelope Control

As soon as an input signal is received the amplified output at Q1 collector is half wave rectified by D1 and smoothed by C1. The result is that the voltage on IC1a's input, which is normally held at the positive supply voltage by R7, falls sharply towards 0V and remains there for the duration of the input signal. IC1a is connected as an inverting Schmitt trigger and its output will normally be at 0V. When the voltage across R7 falls below half of the supply voltage, IC1a's output will immediately switch to 9V. Capacitor C2 couples the positive pulse from the output of IC1a to the input of IC1b.

The two inverting Schmitt triggers IC1b and c are connected together to form a non inverting Schmitt trigger circuit. Normally IC1b's input is held low via RV1 and therefore IC1c's output is at 0 volts. When the output of IC1a switches to 9V, IC1b's input is also switched to 9V via C2. IC1b's output therefore falls to 0 volts and IC1c's output rises to 9V. C2 immediately starts charging via RV1 and the voltage at the input of IC1b falls towards zero. When IC1b input has fallen to approximately half supply voltage its output switches back to 9 volts, and IC1c's output returns to 0V. The length of time for which IC1c's output is at 9V is set by the values of C2 and RV1. This time, the 'sustain' period, is adjustable from zero to approximately 0.5 seconds by RV1.

During the sustain period the base of Q3 is held positive by IC1c's output via D4. C3 initially at 0V, charges via Q3 at a rate set by the attack control RV2. At the end of the sustain period when IC1c's output falls to 0 volts – Q3 is turned off, and C3 discharges via D5 at a rate set by the decay control RV3. The voltage on C3 is buffered by emitter follower Q6, and used to bias the collector load resistor of the output driver transistor Q5. The maximum positive swing of the output stage therefore depends on the voltage across C3 – the envelope control voltage. Whenever the signal from the Spectrum ceases, IC1a's output falls to zero, C3 is discharged via R8 and D3, and the output stage is turned off.

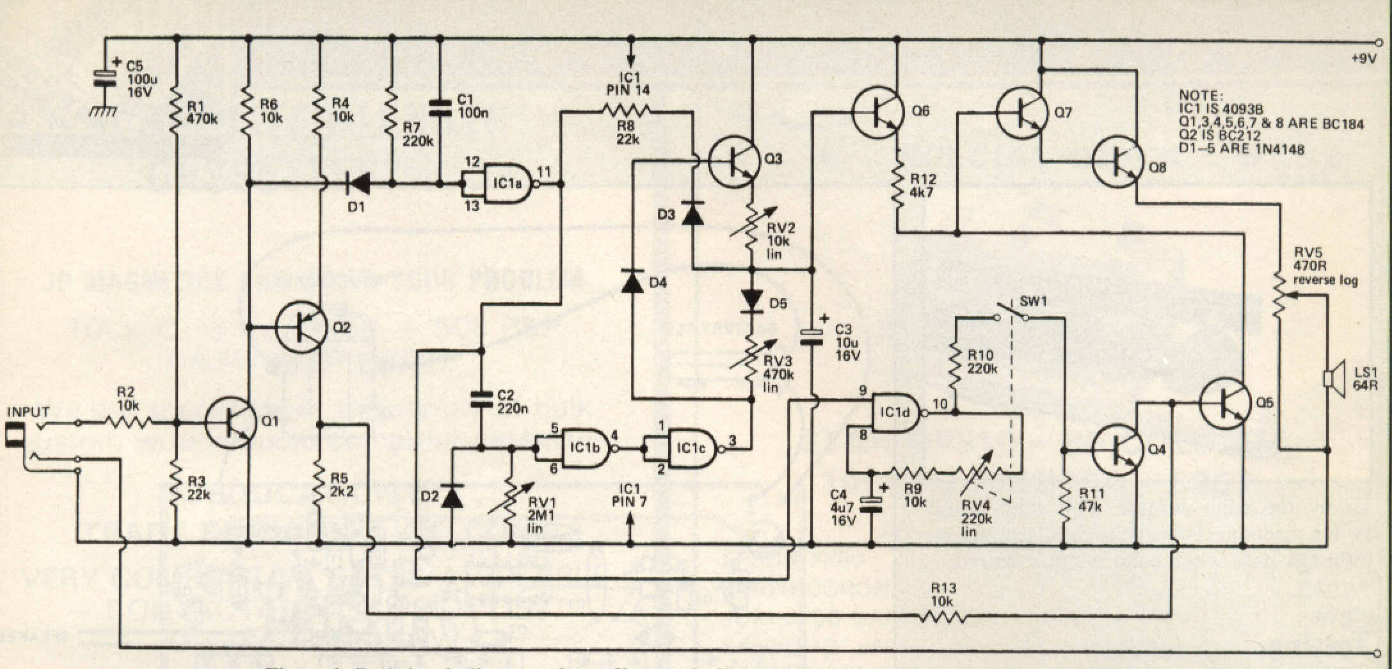


Figure 1. Full circuit diagram of the effects unit. This can be broken down into three sections – an Amplification Stage (Q1, Q2 & Q5), the Envelope Control (IC1a, Q3 & Q6) and the Echo circuitry (Q4 & IC1b-d).

The Amplifier

The Spectrum provides approximately 1 volt peak to peak audio output, from the 'mic' socket via a DC blocking capacitor. The output waveform is a square wave over most of the audio range, with a gradual low frequency roll off. Input transistor Q1 is biased by R1 and R3 to OV4 so that it is normally cut off. Positive half cycles of the input signal added to the bias voltage are sufficient to turn Q1 fully on. Q1 collector swings from 9V down to nearly 0 volts during positive input half cycles, and back to 9 volts as Q1 is cut off during negative input half cycles.

Switching amplifier, Q2, buffers and signal at Q1 collector and feeds output driver transistor, Q5. If the envelope circuitry (Q3, Q4 and IC1) is ignored for a moment Q6 can be assumed to be turned on. R12 forms the collector load for Q5, the driver transistor. Output transistors Q7 and Q8 are connected as a darlington pair operating in emitter follower mode, driving the high level volume control RV5 and the loudspeaker. Maximum output is 7 volts peak to peak across the 64 ohm speaker, this is 200mW for a square wave signal.

BUYER'S GUIDE

We have arranged for Magenta Electronics to supply a complete kit of components for this project.

The price is £13.99 including VAT – postage and packing is 50p extra.

Magenta Electronics
135 Hunter Street
Burton-on-Trent
Staffs
DE14 2ST

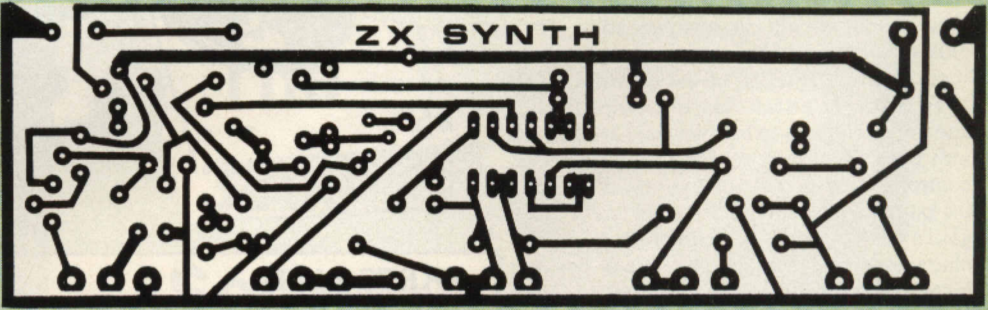


Figure 2. The full size foil pattern of the effects unit's PCB.

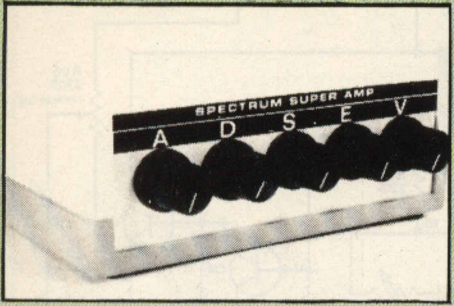
Assembly

The whole circuit is built on a single printed circuit board. Refer to the printed circuit board overlay drawing, and insert and solder the components in the usual order – low profile components first, large components and wire leads last. Note in particular that Q1 must lie flat down to the board to leave room for the switched potentiometer RV4/SW1. Resistor R4 is fitted on end, and R10 is mounted between the board and SW1. Make sure that the electrolytic capacitors, diodes, transistors, and IC1 are the right way round. A socket is recommended for CMOS IC1.

It is important to use the correct type of PCB mounting pots for RV1 – RV5. The assembled board is mounted to the case front panel by the potentiometer bushes. Jack socket SK1 is a switched type with a normally open contact on the sleeve terminal. Alternatively a standard jack socket and a separate toggle switch can be used, with the necessary wiring change. The prototype was built in a snap together verobox with slide in front and rear panels. The loudspeaker is glued inside the top half of the case.

Echo

The two Nand inputs of IC1d are used to control the echo effect. One input is connected to the output of IC1a while the other input along with C4, R9 and RV4, forms an inverting Schmitt trigger oscillator around IC1d. By using the two inputs in this way the oscillator section is switched on whenever an input signal from the computer is present. From IC1d's output the square wave oscillator's signal passes via R10 to the base of Q4. Positive half cycles from IC1d turn on Q4, and remove drive from Q5 base. The output is therefore interrupted at a rate



set by the echo control. The echo on-off switch disconnects the echo oscillator output from Q4 base when echo is not required.

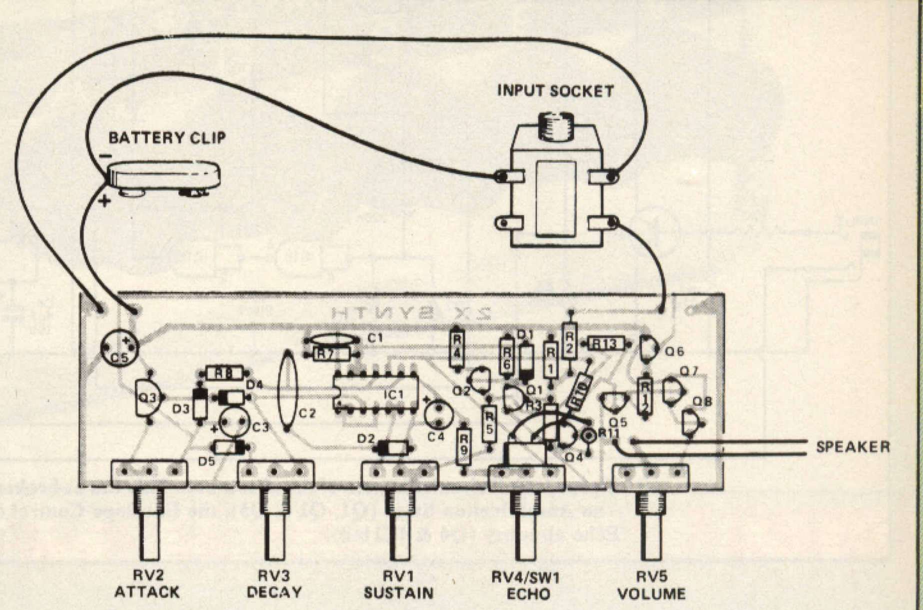


Figure 3. The overlay shows all the component positions and connections to the off-board components.

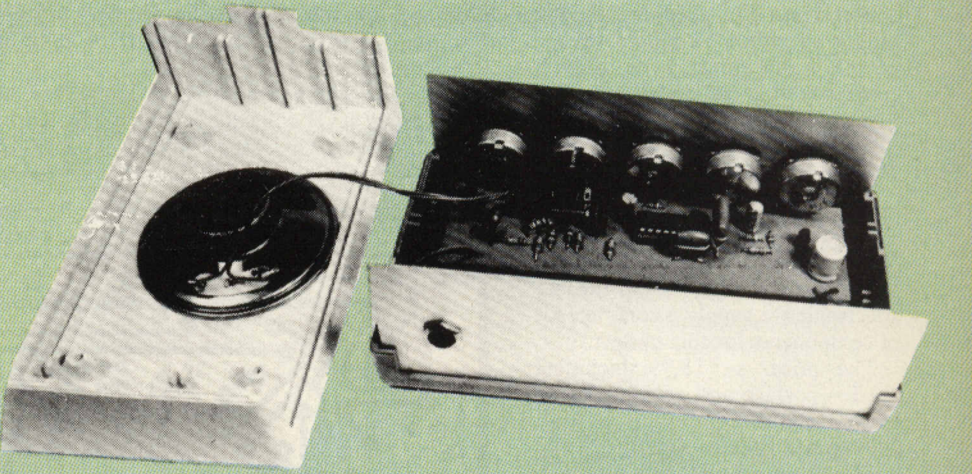
Testing

Fit a PP3 battery to the amplifier and connect the input to your Spectrum 'mic' socket, using the tape in-out lead. Turn the echo control off – set the attack, sustain and volume controls fully clockwise, and the decay control fully anticlockwise. Enter a simple 'Beep' command to produce a 4 second beep at about middle C. The output should be a loud beep that cuts off sharply after about half a second. Reduce the setting of the sustain control and the output beep should cut off in a shorter time. If the sustain control is moved to zero there will be no output because the Beep has been shortened so much that is disappears.

Next ecperiment with the deay control. Set the sustain to maximum and the decay control to halfway. Enter a four second beep – the output should die away steadily. Try different settings of the decay control to get some idea of the range of the control.

To check the attack control set the sustain and decay controls to half way and gradually turn back the attack control. As the attack control setting is reduced the rate of rise of the output signal should be heard to change, giving the sound a more 'gentle' characteristic. Note that the attack period takes place during the sustain time. If the sustain time is too short the output will not have time to rise to its full output before the decay commences.

To test the echo effect set the decay and sustain controls fully clockwise and enter a long beep. Switch on the echo and set the rate as required. At the higher speeds the echo becomes more like a second audio tone modulating the beep, giving some interesting musical, and not so musical, effects. Assuming everything works normally the unit is ready for use. Note that below about 50Hz the effects controls cease to operate. This is because each half cycle re-triggers the sustain circuit. This is a very minor limitation in practice since most applications use much higher frequencies. **E&CM**



PARTS LIST

Resistors – 0.25W 5% carbon film

R1	470K
R2, 4, 6, 9, 13	10K
R3, 8	22K
R5	2K2
R7, 10	220K
R11	47K
R12	4K7
Potentiometers	
RV1	2M2 lin
RV2	10K lin
RV3	470K lin
RV4/SW1	220K lin switched
RV5	470R reverse log

All pcb mounting type 0.2" pin spacing

Capacitors

C1	100n
C2	220n
C3	100u
C4	4u7u
C5	100u
Semiconductors	
D1-5	IN4148
Q1, 3, 4, 5, 6, 7, 8	BC184
Q2	BC212
IC1	4093B CMOS

Miscellaneous

64 ohm miniature speaker, PCB, 3.5mm jack socket with 1 n/o contact, PP3 clip, Case– vero 202-21041, Knobs with marker line – 5 off, Connecting wire.

PRINTERS

This month Nick Clare has assembled details of a variety of printers covering a span of prices.

Of all the peripherals that most computer users purchase, printers, along with disc drives, are likely to require the largest capital outlay. In the case of disc drives, the price for a given amount of storage tends to be much the same no matter from whom the drive is purchased. In the case of printers however, the variation in price for what at first glance may seem models of much the same performance can vary greatly. Fundamentally though the old adage 'you get what you pay for' very much applies to the printer market.

As with most products in the computer market place, there are fundamental trade offs between various aspects of a printer's performance and between these and a particular model's price. One of the major trade offs is between a printer's speed and the quality of printout – the faster the printer the lower the quality. While this is generally true it is possible to identify slow printers with low quality output of Sinclair's ZX printer and to printers capable of almost letter quality output at a fair speed.

A variety of techniques are used to produce a printed output from the very low cost thermal approach, through ribbons and hammers which form the basis of the vast majority of printers to daisy wheel printers at the top end of the market. With the exception of daisy wheels, most output is in the form of a matrix of dots and the number of dots that go to form the matrix play a vital part in determining the ultimate quality of the printed characters.

The other major aspect of a printers specification is the communication protocol adopted. Most printers in their base version will feature a centronics parallel interface – an RS232 option is usually available at a premium of about £40/£50.

Low Cost Models

The lowest cost printer, not surprisingly, is the **Sinclair ZX** printer. Much has been said about this printer – mostly about its slow speed and poor quality – and it is mentioned merely to get the ball rolling. The ZX printer also suffers from the disadvantages that, without a special interface, it can only be used with ZX81 or the Spectrum.

Another printer designed for the ZX computer is the **Dean Electronics** Alphacom 32. This printer uses thermal paper, as does the Sinclair printer, but uses a 4.5" wide roll that prints 32 characters per line at a rate of two lines per second. The printer features a seld test mode and a paper advance facility.

The Alphacom 32 costs £99.95 inclusive of VAT and postage and packing. One 25m roll of paper is also included in the price.

Forever Amber

At £89.70 plus £3.50 p&p, the Amber 2400 matrix printer is another low cost solution to the problem of hard copy output. This printer uses plain paper rolls and print hammers plus an ink ribbon. The output looks much the same as that produced by cash dispenser machines (not the pound notes but the acknowledgement slips!) being very legible if not elegant.

The printer provides both parallel and serial interfaces and Amber can supply connecting leads in order that the printer can be used with the BBC Micro, the Acorn Atom, the 101, the Dragon and Atari 400/800 as well as the New Brain. A who's who of the popular computers available at present and the company may be prepared to discuss interfaces for other machines.

ALPS Mechanisms

In the £100-£200 price range about the only printers to be found are based around the ALPS printer mechanism. Both the Oric and Tandy printers are built around the four colour printing/plotting unit. These printers are capable of producing reasonable quality text along with intricate graphic patterns.

The Tandy unit can successfully be used in conjunction with the BBC micro as well as with Tandy's machines.

If the mixture of text and graphics is required, these printers are an ideal choice.

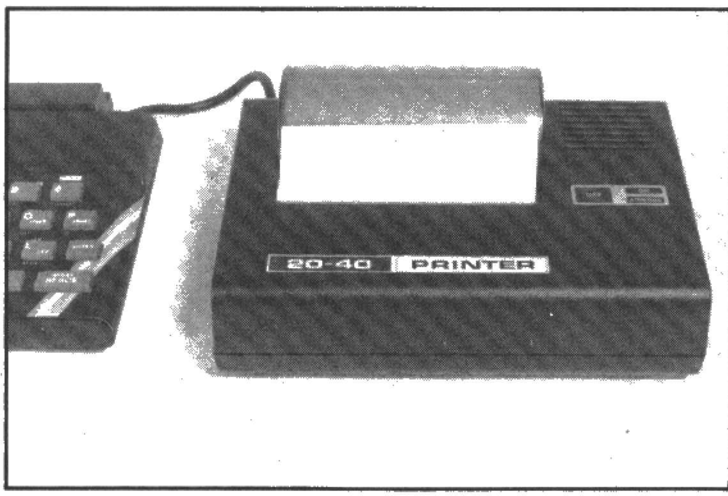
Above £200

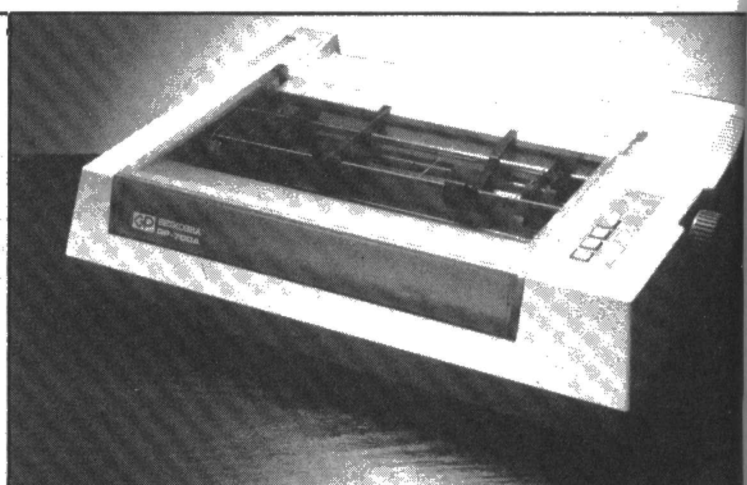
Once the £200 barrier is broken, the choice of printers begins to widen considerably. The **Seikosha** range of printers, available from Intelligent Interfaces, have a VIC 20 graphics printer that just breaks the £200 barriers by £5 although VAT must be added to this price. It uses a uni-hammer printing method to produce a 6 × 7 dot matrix. Printing speed is 30 char/sec and the maximum width is 80 columns.

The GP-100A is a version of the VIC printer that features a standard centronics compatible interface (RS232 as an option). The price of this printer is £215 plus VAT.

At £425 + VAT the GP-700A is another printer from Seikosha – a printer that can produce colour output. The added impact of colour output will be worth the extra cost in many applications that mix text with graphics or where various sections of text need to be highlighted.

Intellect interfaces also stock a range of other printers and their





price list (which includes a variety of cables and buffers) is worth obtaining.

Another source of low-cost dot matrix printers is **Geveke Electronics.** The range of printers from the Company includes the GLII/GSII 80 column/100 character per second model. This is a tractor feed printer that provides either serial or parallel interfaces. It incorporates a one line buffer and uses a nine needle print head to produce its output.

Geveke also supply a number of more expensive printers with the star of their range being the Diablo ink jet colour printer.

The Form Of Flattery

Some of the most imitated printers on the market are those produced by Epson. **Norbain Micro Ltd.** stock the two main stays of the Epson range – the RX80 at £298 plus VAT and The FX80 at £438 plus the VAT.

The FX80 follows established Epson design philosophy and incorporates a variety of type styles in ROM as well as RAM memory for storing up to 256 user defined characters or alternatively a 3K buffer.

The RX80 is a slimmed down version of the FX80 omitting such features as the in-built RAM.

Epson printers have a reputation for reliable operation – something that is of prime importance to many users – the *E&CM* office was brought to a halt when one printer went down recently.

Another range of printers worth consideration if a dot matrixs design is required are the models produced by **Mannesman Tally.**

These include the 80 column, 80cps MT80 with dot addressable and line graphics through to the MT180 capable of producing 160cps flat out which slows down to 40cps when the machine is producing letter quality output.

The range is available from CK computers who also stock other ranges of printers.

Another good source of printers and accessories is **Thame Systems Ltd.** They stock a wide range of printers including models by Axiom, Brother, NEC and Toshiba. Looking through the Company's catalogue should reveal a printer to suit most applications.

If the ultimate quality is demanded from a printer and speed is not of the essence there can be no doubt that a daisy wheel printer is the best choice. You have to pay for the quality though and may well have to spend in excess of £1000.

If this sort of printer is what you require **APT** can help with a range of sophisticated high quality printers.

See Before Buying

With printers, more than most peripherals, its a good idea to see the printer in action before making a purchase. Print quality, noise levels and speed are all major factors in choosing a printer and are best assessed by observing the machine in action. **E&CM**

Dean Electronics
Glendale Park
Fernbank Road
Ascot
Berkshire

Amber
Central Way
Walworth Industrial Estate
Andover
Hampshire

Intelligent Interfaces
18 Central Chambers
Wood Street
Stratford-on-Avon
Warks
CV37 6JD

Geveke Electronics
RMS House
Vale Farm Road
Woking
Surrey
GU11 DW

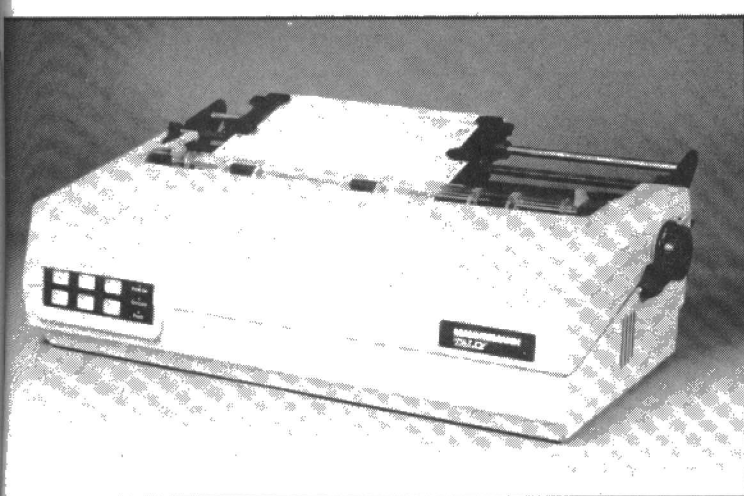
Norbain Micro Ltd.
Norbain House
Boulton Road
Reading
Berks
RG2 0LT

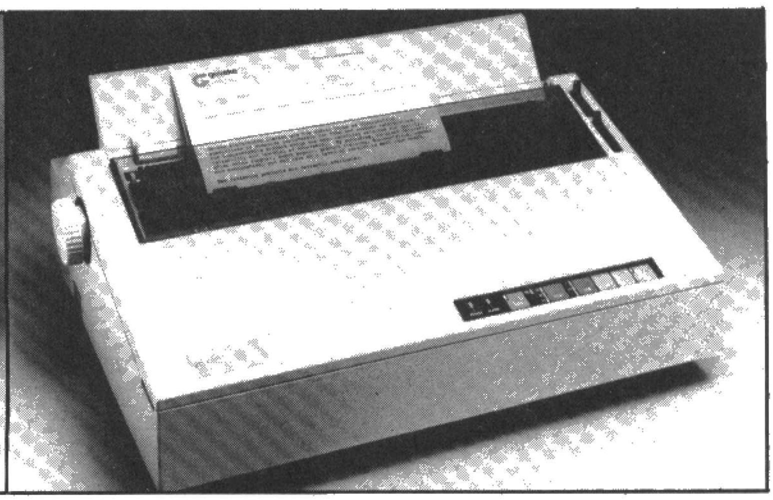
CK Computers Ltd.
6 Devonia House
High Street
Weston-Super-Mare

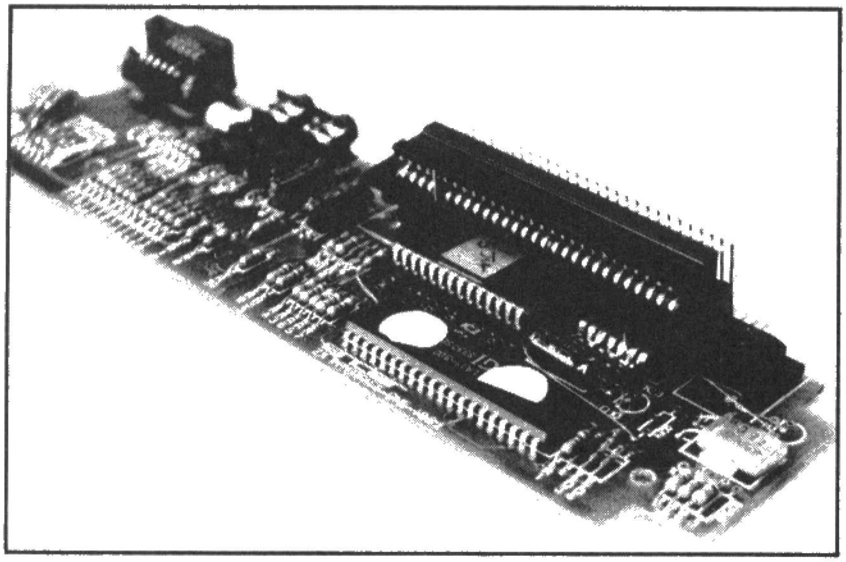
Thame Systems
Thame Park Road
Thame
Oxon
OX9 3XD

APT
2-4 Canfield Place
London
NW6 3BT

From left to right the pictures show Dean Electronic's Alphacom 32, Seikosha's GP 700-A Colour Printer, the MT180 from Mannesman Tally and one of Geveke's range of Printers.







MICRODRIVE UPDATE

THE HARDWARE

Last month our report on the microdrive detailed the additional commands supported by a Spectrum + Interface 1 + Microdrive combination and the form of the various commands concerned with loading and saving data with the Microdrive. This month we've had a chance to look at the hardware of the drive and have discovered a few things that might cause problems in the future.

With any review of a new product a degree of caution must be exercised. If a manufacturer supplies a reviewer with a pre-production version of a new piece of equipment, it would clearly be unfair to criticise the standard of construction if any shortcomings in this area would not be apparent on the production version.

In the case of the Microdrive however, to the best of our knowledge, the drive and interface supplied to us were exactly the same as those being sent out to the lucky people at the top of Sinclair's waiting list.

The Microdrive itself looks to be a well constructed, sturdy unit that has been production engineered to a high standard. Thorn EMI should be able to churn out reliable drives at a fair old rate at an ex-factory price that will leave plenty of change from the £50 asking price of the drive.

Interface 1 is however an entirely different proposition. Opening up the Interface reveals a board with a surprising amount of discrete circuitry. Much of this is assumed to be responsible for the RS232 interface and for the networking circuitry. To find such a number of resistors and transistors was however a surprise and we could not help but think that a DIL resistor package or two plus a transistor array would have reduced costs and added to the unit's reliability.

The other major surprise was the number of wire links and ICs with flying wires attached to their legs. Clearly there were a number of mistakes in the tracking out of the PCB. In the main this sort of bodging is just about acceptable, in one case though, one end of a flying resistor was soldered to a track that was so thin that it was expecting rather a lot for it to stay in place if the interface were subjected to anything but the gentlest of handling. In fact after dropping our Interface, not deliberately we hasten to add, said resistor became detached and meant that the Drive failed to read anything from the data cassette.

In view of the fact that the PCB is clearly not in its final form the fact that the software is in EPROM rather than ROM is not so surprising.

It is difficult to see why, when Sinclair have taken over a year to provide the system, that when launched it should be as unfinished as it evidently is.

These 'bodges' also explain why the major factor limiting the production of microdrive systems is the rate at which Interface 1's can be produced – clearly manufacturing a board with so many flying leads must be a nightmare.

Connection Details

Our examination of the Microdrive lead to the two circuit diagrams shown. these are our best guess as to the location of the read and write lines to the drive. *E&CM's* tame hardware/software design engineer's to whom we lent our drive were of the opinion that it should not be too difficult to hook the drive up to other computers – Sinclair won't like it but thousands of Oric/Dragon/BBC owners would be clamouring for any unit that would allow the drive to be used with their computer.

Summing Up

We cannot help but take our hats off to Sir Clive and his development team. The Microdrive for all its shortcomings, both in hardware design and in the patchy quality of the manual, represents a major breakthrough in low-cost mass storage systems.

We can only hope that the Microdrive is reliable in use for, with the likely level of sales, any company would be hard pushed to deal with the problems resulting from an unreliable product.

We think the drive is, if not the best thing since sliced bread, certainly yet another milestone in Sinclair's illustrious history.

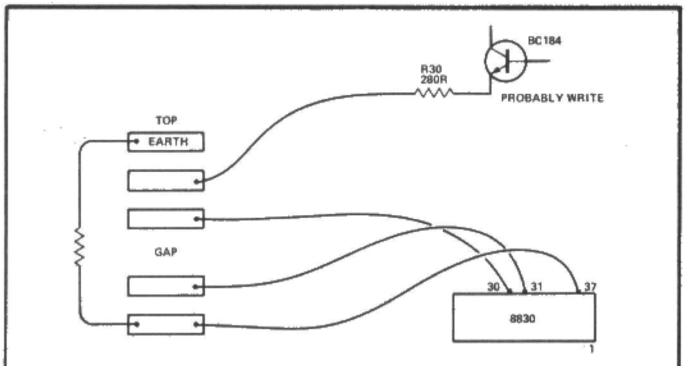


Figure 1. Connections to the topside of the Microdrive's edge connector.

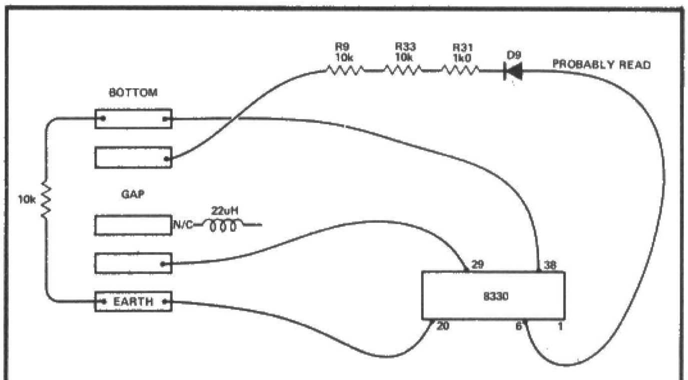


Figure 2. The lower side connections.

If you're still staggering through the computer jungle and not getting sensible answers to your questions, we have some good news:

Now you can turn to **professional** people who are capable of giving you sound advice on practically all aspects of popular computing.

They all have one thing in common:

They are **COMPUTERS FOR ALL** dealers.

A Computers for All dealer is different from the normal Computer retailer.

Not surprisingly he won't try to sell you things like cameras or cosmetics; stationery or sealing wax.

He will, however, be capable of answering sensibly almost any question you have on computers and computing, and have readily available a wide range of popular computers, hardware, software, books, and peripherals.

So why not call in at your local **COMPUTERS FOR ALL** dealer today? He can lead you in the direction you want to travel.

The shops where people matter:

AVON
JADE COMPUTERS
Coombend, Radstock, Bath. 0761 32570.
MERCATOR COMPUTER SYSTEMS
3 Whiteladies Road, Clifton, Bristol. 0272 731079.
MOBILE MICROS
2 Castle Street, Thornbury, Bristol. 0454 418383.

BERKSHIRE
KENNETH WARD COMPUTERS
Verve House, London Road, Sunningdale. 0990 25025.

CHANNEL ISLANDS
MEGA LTD.
7 Anley Street, St. Helier, Jersey. 0534 72263.

CHESHIRE
THE COMPUTER CENTRE
68 Chestergate, Macclesfield.

CORNWALL
COMPUTAVISION
4 Market Street, St. Austell. 0726 5900.
FAL-SOFT COMPUTERS
8 St. George's Arcade, Falmouth TR11 3DH. 0326 314663.

DEVON
A & D COMPUTERS
"Computerland", 6 City Arcade, Fore Street, Exeter. 0392 77117.
BITS & BYTES
44 Fore Street, Ilfracombe, N. Devon EX34 9JD. 0271 62801.
COMPUTER SYSTEMS (TORBAY)
Pump Street, Brixham. 08045 6565/6.
CRYSTAL COMPUTERS
209 Union Street, Torquay. 0803 22699.
SYNTAX LTD
Midhurst, Grenofen, Tavistock. 0822 2392.

DORSET
DENSHAM COMPUTERS
329 Ashley Road, Parkstone, Poole.

ESSEX
AKHTER INSTRUMENTS
Unit 19, Arlinghyde Estates, South Road, Harlow. 0279 412639.
COMPUTERS FOR ALL
72 North Street, Romford. 0708 752862.

HANTS
MICRO VIDEO STUDIOS LTD.
17 Turks Street, Alton. 0420 82055.

HERTS
VIDEO CITY
45-47 Fishers Green Road, Stevenage. 0438 53808.
THE COMBINED TRADING CO.
10 & 11 Salisbury Square, Old Hatfield. 07072 65551.
THE ROYSTON MICROCENTRE
1 John Street, Royston. 0763 42622.

KENT
APHROS SOFTWARE CO.
47 Hawley Square, Margate, Kent. 0843 294699.
MEDWAY COMPUTERS
141 New Road, Chatham. 0634 826080.
MICRO MAGIC
128 Erith Road, Bexleyheath. 0322 523052.
THE DATA STORE
43 Shepperton Road, Petts Wood. 0689 26698.
THE MICRO SHOP
11 The Pantiles, Tunbridge Wells. 0892 27691

LANCS
P.C.S.
39 Railway Road, Darwen. 0254 776677.
4 MAT COMPUTING
67 Friargate, Preston.
THE HOME COMPUTER CENTRE
40 King Street, Blackburn. 0254 671316.

LEICS
DIMENSION
27-29 High Street, Leicester. 0533 57479.

LONDON
KAYDE HOME COMPUTERS
1 Station Approach, New Eltham, London SE9. 01-859 7505.
KELLY'S COMPUTERMARKET
227 Dartmouth Road, Sydenham, London SE26 4QY. 01-699 4399/6202.

MIDDLESEX
SCREENS MICROCOMPUTERS
6 Main Avenue, Moor Park, Northwood. 09274 20664.
TWILLSTAR COMPUTERS
17 Regina Road, Southall. 01-574 5271.

N. IRELAND
D. V. MARTIN LTD.
13 Bridge Street, Belfast, BT1 1LT. 0232 226434.
MCLAUGHLIN ELECTRONICS
44 Carlisle Road, Londonderry. 0504 65002.

NORTHAMPTONSHIRE
LEISURETIME
13 The Friary, Grosvenor Centre, Northampton. 0604 36726.

OXFORDSHIRE
SCIENCE STUDIO
7 Little Clarendon, Oxford OX1 2HP. 0865 54022.

SHETLANDS
TOMORROW'S WORLD
Esplanade, Lerwick, ZE1 0LL. 0595 2145.

SUFFOLK
BECCLES COMPUTERS
5 The Ridings, Worlingham, Beccles. 0502 715061.

SURREY
ANIROG COMPUTERS
8 High Street, Horley. 02934 2007.
COMPUTASOLVE
8 Central Parade, St. Marks Hill, Surbiton. 01-390 5135.

SUSSEX
THE COMPUTER CENTRE (BMS) LTD.
37d & 37e Robertson Street, Hastings, East Sussex. 0424 439190.

S. WALES
AUTOMATION SERVICES (S. WALES)
3 Wermeys Road, Penysfai, Bridgend. 0656 720959.
DAN EVANS (BARRY) LTD.
81 Holton Road, Barry, South Glamorgan. 0446 734242.
MORRISTON COMPUTER CENTRE
37 Clase Road, Morriston, Swansea SA6 8DS. 0792 797572.
STEVE'S COMPUTER CO. LTD.
Castle Arcade, Cardiff, South Glamorgan. 0222 41905.

TYNE & WEAR
THE COMPUSHOP
10 Newgate Centre, Newcastle-upon-Tyne NE1 5RE. 0632 618673.

WARWICKSHIRE
IMPULSE MICRO SYSTEMS LTD.
6 Central Chambers, Cooks Alley, Wood Street, Stratford-Upon-Avon. 0789 295819.

WEST MIDLANDS
CALISTO COMPUTERS LTD.
119 John Bright Street, Birmingham B1 1BE. 021-632 6458.
JBC
200 Earlsdon Ave. North, Earlsdon, Coventry. 0203 73813.

WORCESTERSHIRE
DEATH VALLEY COMPUTERS
P.O. Box 54, Worcester WR2 6QA. 0905 640400.
EVESHAM COMPUTER CENTRE
Crown Court Yard, Bridge St., Evesham. 0386 48635.

YORKSHIRE
COM-TEC
6 Eastgate, Barnsley, South Yorkshire. 0226 46972.

COMPUTERS FOR ALL